MIT/LCS/TR-548

A TIMING ANALYSIS OF LEVEL-CLOCKED CIRCUITRY

Alexander T. Ishii
Charles E. Leiserson

September 1992

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE

A Timing Analysis of Level-Clocked Circuitry

5. FUNDING NUMBERS

6. AUTHOR(S)

Ishii, A. T., Leiserson, C. E.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

MIT, Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

8. PERFORMING ORGANIZATION REPORT NUMBER

MIT/LCS/TR-548

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

N00014-91-J-1698

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper presents an algorithm for verifying proper timing in VLSI circuits where latches are controlled by the levels (high or low) of the controlling clocks rather than the transitions (edges) of the clocks. Such level-clocked circuits are frequently used in MOS VLSI design. A level-clocked circuit is modeled as a graph $G = (V, E)$, where V consists of components—latches and functional elements—and E represents intercomponent connections. The algorithm verifies the proper timing of a circuit in worst-case $O(|V||E|)$ time and $O(|V| + |E|)$ space.

Our analysis decouples the problem of generating timing constraints from the problem of efficiently checking them. We show how various "base step" functions can be used to provide sufficient conditions for a circuit to operate properly, and we provide a new base step function which is less pessimistic than those used in previous timing verifiers, yet correctly handles timing constraints that are "cyclic" or extend across the boundaries of multiple clock phases or cycles. The base step function is used to derive a "computational expansion" of the circuit from which a collection of simple linear constraints are derived. These constraints can be efficiently checked using standard graph algorithms.

14. SUBJECT TERMS

VLSI systems, level-clocking, timing constraints, timing analysis, timing verification, computational expansions, delta-constraints, formal modeling, graph algorithm applications, algorithmic tehniques

15. NUMBER OF PAGES

32

CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

A Timing Analysis of Level-Clocked Circuitry

Alexander T. Ishii
Charles E. Leiserson

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

July 22, 1992

Abstract

This paper presents an algorithm for verifying proper timing in VLSI circuits where latches are controlled by the levels (high or low) of the controlling clocks rather than the transitions (edges) of the clocks. Such level-clocked circuits are frequently used in MOS VLSI design. A level-clocked circuit is modeled as a graph $G = (V, E)$, where V consists of components—latches and functional elements—and E represents intercomponent connections. The algorithm verifies the proper timing of a circuit in worst-case $O(|V||E|)$ time and $O(|V| + |E|)$ space.

Our analysis decouples the problem of generating timing constraints from the problem of efficiently checking them. We show how various "base step" functions can be used to provide sufficient conditions for a circuit to operate properly, and we provide a new base step function which is less pessimistic than those used in previous timing verifiers, yet correctly handles timing constraints that are "cyclic" or extend across the boundaries of multiple clock phases or cycles. The base step function is used to derive a "computational expansion" of the circuit from which a collection of simple linear constraints are derived. These constraints can be efficiently checked using standard graph algorithms.

Keywords: VLSI systems, level clocking, timing constraints, timing analysis, timing verification, computational expansions, delta-constraints, formal modeling, graph algorithm applications, algorithmic techniques.

[0]This research is supported in part by the Defense Advanced Research Projects Agency under Contracts N00014-87-K-0825 and N00014-89-J-1988, and Grant N00014-91-J-1698. Charles Leiserson is supported in part by an NSF Presidential Young Investigator Award with matching funds provided by AT&T Bell Laboratories, IBM Corporation, and Xerox Corporation.

A preliminary version of this paper appears in *Advanced Research in VLSI: Proceedings of the 6th MIT Conference*, 1990.

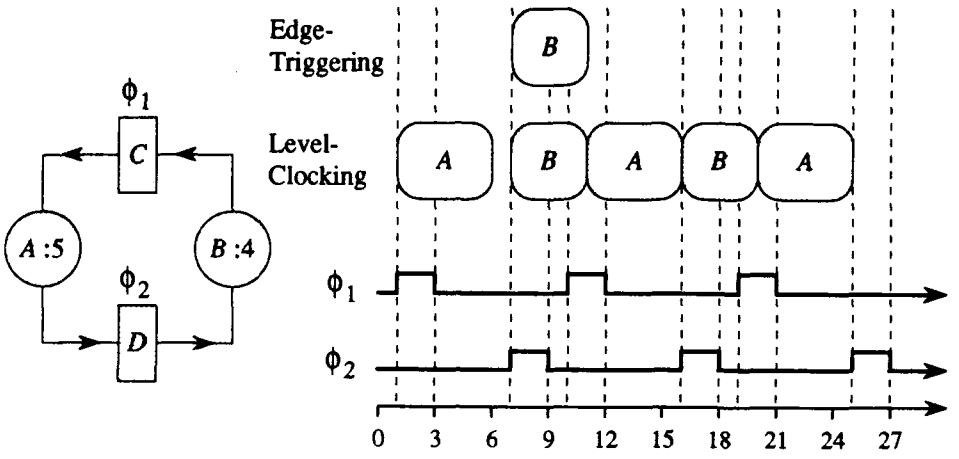

Figure 1: An abstract representation of a level-clocked circuit is shown, along with its associated clocking waveforms. Each circle represents a functional element (*e.g.*, block of combinational logic) which has associated with it a label and a propagation delay. Each rectangle in the figure represents a level-clocked latch which has associated with it a label and a controlling clock waveform.

1 Introduction

MOS/VLSI technology has popularized a methodology of clocking based on level-clocked latches instead of the more traditional edge-triggered latches used, for example, in TTL [20] design. The popularity of level-clocking arises from the simplicity with which a level-clocked latch can be implemented in MOS technologies: a single transistor can suffice [6, 13]. Unfortunately, level-clocking methodologies make the problem of determining whether a circuit is properly clocked a difficult one, because changes in the output of a latch need not closely correspond to transitions in its clocking waveform. In contrast, the output of an edge-triggered latch changes only on a transition of its clock, and consequently, the propagation of computation through the circuit can be more easily tracked.

To illustrate the basic concepts of level-clocked circuit operation, consider the circuit depicted in Figure 1. (A similar example is discussed in [6, p. 334].) Each circle in the figure represents a functional element (*e.g.*, block of combinational logic) which has associated with it a label and a propagation delay. The propagation delay of a functional element specifies the "settling" time required for the output to assume its correct value after an input changes. Until it settles, the value of the output is considered to be undefined. Each rectangle in the figure represents a level-clocked latch which has associated with it a label and a controlling clock. While the clock for a latch is high, the output of the latch is equal to its input. When the clock changes to low, the latch stores the value of its input and outputs this value until the clock changes back to high.

A natural question to ask is whether the circuit of Figure 1, with the propagation delays and clocking waveforms shown, computes properly. For example, after suitable initialization of latch outputs at start-up, do all latches always hold well-defined values? It might appear that the answer is no, because of the following fallacious reasoning. At time 12, the input of Latch C should be the result of applying the function computed by B to the output of Latch D at time 9. Thus, B may have to start a computation at time 9 and finish by time 12, *i.e.*, finish in 3 time units, but its propagation delay is 4 time units, which is too long.

This reasoning is improper because the computation of B can always begin before time 9. To see this, look back at the output of Latch C, which we presume must have a proper value at time 3 when clock ϕ_1 goes low. At time 8 the output of A has settled, and since clock ϕ_2 is high, this value passes immediately to the output of Latch D. Thus, since the output of C can not change between times 3 and 10, B can always begin its computation by time 8 instead of time 9, as in the fallacious analysis, and the computation must complete successfully by time 12. Observe that Latch D transmits a value at a time distinct from any transition of its clock. This is in contrast to the situation where all latches are edge-triggered, in which case the time at which a latch transmits a value corresponds directly to a transition of its clock.

The circuit of Figure 1 illustrates the most basic type of "timing constraint" that must be met to ensure proper circuit operation. In general, computations are constrained to occur between the rising edge of one

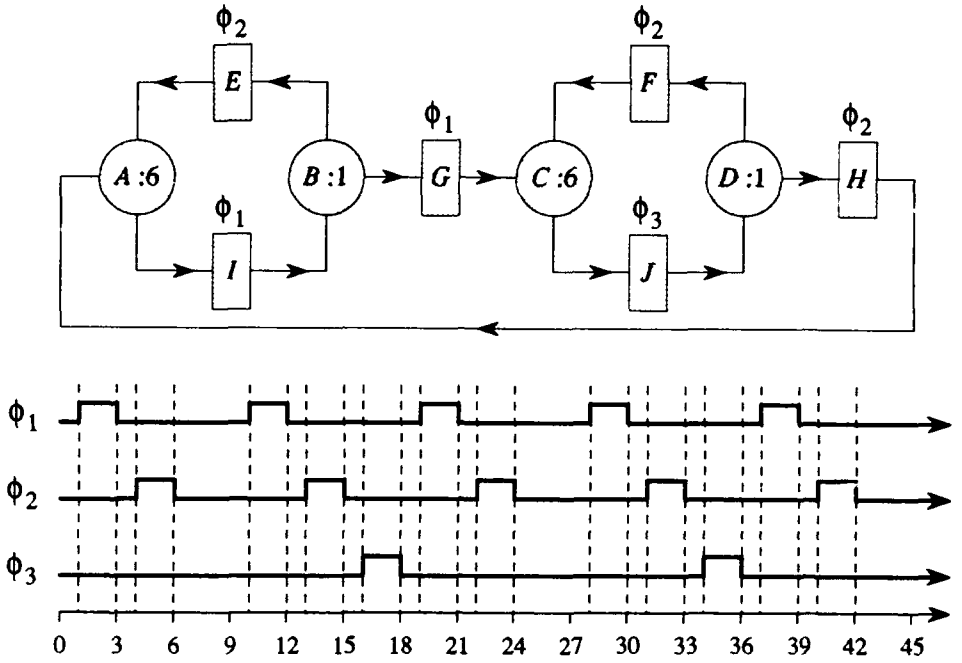

Figure 2: A circuit which demonstrates some of the subtleties of level-clocked circuitry.

clock and the next falling edge of another. For example, between the rising edge of ϕ_1 at time 1 and the falling edge of ϕ_2 at time 9, A must be able to take a new input and compute a new output. Thus, the propagation delay 5 of A must be less than the amount of time between the rising edge of ϕ_1 and the falling edge of ϕ_2, *i.e.*, less than 8. Similarly, the propagation delay 4 of B must be less than the amount of time between the rising edge of ϕ_2 at time 7 and the falling edge of ϕ_1 at time 12, *i.e.*, less than 5. These two constraints on the propagation delays of functional elements are examples of the *delay constraints* that have been widely recognized in the literature [1, 4, 6, 8, 9, 11, 14, 15, 16, 17, 18, 21, 22].

The circuit also illustrates the *scheduling constraints* that have been considered by previous timing analyses. Intuitively, scheduling constraints require that functional elements not begin using their inputs before the inputs are in fact available. For example, if A receives a new input just as ϕ_1 falls at time 3, then the new output of A, which presumably propagates through latch D between times 7 and 9, is not ready until time 8. Thus, the computation of B is constrained not to begin until time 8 which in turn implies that the output of B will not be ready until time 12, when ϕ_1 falls. Observe that since the output of B is ready by the falling edge of ϕ_1, we have in effect found a consistent "schedule" that has A computing between the fall of ϕ_1 and one time unit before the fall of ϕ_2, and has B computing between one time unit before the fall of ϕ_2 and the fall of ϕ_1. If the propagation delay of A were 6, instead of 5, no consistent schedule would be possible, since the computation of A would have to start before the fall of ϕ_1. The impossibility of a consistent schedule would constitute a scheduling constraint violation.

The more complex circuit depicted in Figure 2, demonstrates some of the subtleties that can arise in level-clocked circuitry. For example, notice that when ϕ_1 goes high at time 10, functional element B begins a computation whose result must "flow through" latch G before ϕ_1 goes low at time 12. Thus, the circuit contains a delay constraint that occurs between transitions that are part of the same clock. In addition, the time between the rise of ϕ_1 at time 10 and the fall of ϕ_3 at time 18 must be at least the propagation delay 6 of C *plus* the propagation delay 1 of B, rather than just the propagation delay of C. As another example, notice that along the path $F \rightarrow C \rightarrow J$, there is an apparent delay constraint violation. Specifically, there are only 5 time units between the rise of ϕ_2 at time 13 and the fall of ϕ_3 at time 18, while the delay of C is 6 time units. In fact, since ϕ_3 is never high between time 6 and time 15, the output of D during times 13 thru 15 must be the same as it was at time 6. Thus, no new computation by C can begin between times 13 and 18, so the violated delay constraint was in fact "fictitious."

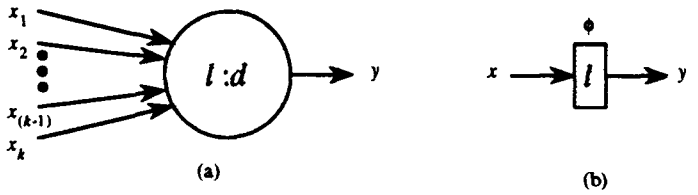

Figure 3: (a) A functional element has some finite number k of inputs x_1 through x_k, a single output y, an associated k-input function f, and a propagation delay d. (b) A level-clocked latch has a single input x, a single output y, and an associated digital clock ϕ.

In the literature, several attempts have been made to develop timing analytical techniques for level-clocked circuitry, as well as to develop algorithms and heuristics to perform analysis automatically [1, 4, 8, 9, 11, 14, 15, 16, 17, 18, 21, 22]. These authors have addressed both delay and scheduling constraints in their timing analyses. They have also provided algorithms that are well suited to the circuits for which they were developed. In general, however, previous timing analysis methods have either applied only to specific clocking disciplines [1, 9, 11, 14, 15, 16, 21] or have checked scheduling constraints by using some type of iterative approximation or relaxation technique to verify the existence of some consistent schedule [1, 4, 8, 11, 17, 18, 22].

While working well in practice, the iterative approximation and relaxation techniques used by previous timing analysis systems are not guaranteed to run in polynomial time. These techniques all fall prey to pathological worst cases where each successive approximation, or relaxation, moves the analysis only some small increment toward the desired final solution. In such worst cases, the running time of the analysis can change drastically in response to a very small change in the circuit being analyzed. For example, simply changing the delay of a functional element from 1 nanosecond to 999 picoseconds might cause over an order of magnitude change in the running time of the analysis.

In this paper, we present a polynomial-time algorithm to determine whether a given level-clocked circuit operates properly. Our algorithm can handle arbitrarily complex clocking disciplines, and verifies the proper operation of a circuit in worst case $O(|V||E|)$ time and $O(|V| + |E|)$ space. If circuit components have bounded fanout, then the algorithm runs in $O(|V|^2)$ time. In addition, our algorithm can identify certain types of fictitious delay constraints, and thus is less pessimistic than previous methods.

Our algorithm is based on an analysis technique called computational expansion, which provides a succinct set of provably sufficient conditions for the proper operation of a level-clocked circuit. The computational expansion is in turn based on a choice of "base step" function, which encapsulates sufficient conditions for the circuit to operate properly. We provide one such base step function that subsumes the timing constraints considered by others. Using the computational expansion, we derive a set of sufficient conditions that can be reduced to a collection of simple linear constraints. These constraints can then be checked using standard polynomial-time graph algorithms, and thus our algorithm avoids the potential for extreme worst-case running times which are associated with iterative approximation techniques.

The remainder of this paper is organized as follows. Section 2 gives our formal model for level-clocked circuits. Section 3 defines the concept of a computational expansion of a circuit, that is used in Section 4 to derive sufficient conditions for proper circuit operation. Section 5 examines methods for checking the sufficient conditions, and presents an algorithm that verifies whether a circuit operates properly over some finite interval of time. Section 6 contains our principal contribution: a polynomial-time algorithm for verifying the proper operation of circuits that use an arbitrary periodic set of clocks. Section 7 presents some concluding remarks.

2 Level-Clocked Circuits

In this section we present the formal models upon which our timing analysis algorithms are based. Mathematical definitions are given for functional elements, level-clocked latches, level-clocked circuits, and proper circuit operation. Intuitive descriptions are provided where appropriate.

A *functional element* has some finite number of inputs $x_1, x_2, \ldots, x_k$, a single output y, a k-input function f, and a propagation delay d, as shown in Figure 3(a). The value of the output y at time t is given by the

equation:

$$y(t) = \begin{cases} f(x_1(t), x_2(t), \ldots, x_k(t)) & \text{if } x_i \text{ is stable for all } i = 1, 2, \ldots, k \\ & \text{over the interval } [t-d, t], \\ \perp & \text{otherwise.} \end{cases} \quad (1)$$

Intuitively, a functional element is a block of combinational logic whose output is some function f of its inputs. The propagation delay d of the functional element is a "settling" time that indicates the amount of time required, after an input changes value, for the output to assume its correct value. The *invalid* value $\perp$ indicates that the output has no well-defined value. An input is *stable* over an interval of time if it assumes a constant valid value over the interval. By definition, a stable input is constant. A constant input need not be stable, however, since an input could be constant with the invalid value $\perp$.

There are two features of equation 1 that should be noted. First, if an input changes value, the output immediately takes on the value $\perp$ and does not become valid until a time equal to the propagation delay d after the change in the input. Thus, the "minimum" propagation delay, or "contamination" delay, of functional elements is assumed to be 0, and d in fact represents the "maximum" propagation delay of a functional element. Second, if any input is $\perp$ at time t it is not stable at time t, by definition, and thus the output must be $\perp$. There are functional elements, such as a common MOS NOR gate, where a changing or undefined input does not necessarily imply an undefined output. Our algorithms do not directly exploit this aspect of such functional elements.

Functional elements can be used to represent more general circuit components, much as ideal electrical components, such as ideal resistors, capacitors, and inductors, are used to model real physical devices [2]. For example, a circuit component with multiple outputs can be represented with several one-output functional elements. As another example, a circuit component whose propagation delay varies with the input can be represented with a zero-delay functional element, each of whose inputs is the output of a functional element that computes the identity function and whose propagation delay is the input-to-output propagation delay of the original functional element.

In order to simplify the explanation of our algorithms, we assume that clock waveforms always have well-defined values. Formally, a *clock* ϕ is a mapping from $\mathbb{R} \cup \{-\infty\}$ to {HIGH, LOW}, such that the set $\{t : \phi \text{ has value HIGH at time } t\}$ is a set of nonoverlapping closed intervals, and ϕ changes value only a finite number of times during any finite interval. Observe that the set $\{t : \phi \text{ has value LOW at time } t\}$ is a set of nonoverlapping open intervals, and thus, when ϕ changes value from HIGH to LOW, there exists a well defined last moment in time when ϕ has value HIGH, but no well defined first moment when ϕ has value LOW. Similarly, when ϕ changes value from LOW to HIGH, there exists a well defined first moment in time when ϕ has value HIGH, but no well defined last moment when ϕ has value LOW. This definition is a somewhat arbitrary convention, which has been chosen for the sake of descriptional brevity. A more general model can be found in [10].

A *level-clocked latch* has a single input x, a single output y, and a controlling clock ϕ, as shown in Figure 3(b). The value of the output y at time t is given by the equation:

$$y(t) = \begin{cases} x(t) & \text{if } \phi(t) = \text{HIGH} \\ y(t_{\phi\text{high}}) & \text{if } \phi(t) = \text{LOW and} \\ & t_{\phi\text{high}} = \sup\{t' \le t : \phi(t') = \text{HIGH}\} \end{cases} \quad (2)$$

We generally refer to a level-clocked latch as simply a *latch*. While the clock for a latch has value HIGH, the output of the latch is equal to its input. When the clock changes value to LOW, the latch stores the value of its input at the "last moment" when the clock had value HIGH, and outputs this value until the clock changes value back to HIGH. The propagation delay of a latch is assumed to be zero. Latches with nonzero propagation delays can be modeled by combining zero-delay latches with "padding" functional elements that compute the identity function.

Functional elements and level-clocked latches are the two types of *components* that level-clocked circuits are constructed from. A *level-clocked circuit* is a directed graph $G = (V, E)$, where V is a set of components consisting of functional elements and level-clocked latches, and $(u, v) \in E$ if the output of u is an input of v. We assume without loss of generality that each component has exactly one input edge for each of its inputs. (Any bus-like structures where multiple components drive a single wire can be modeled by a functional element with an input for each component that can drive the bus, and an associated function that can "resolve" bus conflicts.)

For convenience, we generally refer to a component and its output interchangeably. In particular, we often refer to the output of a component by making reference to the component itself. Both a component v and its output are said to *stabilize* at time t if the output of v changes to a valid value at time t. Similarly, both v and its output are said to *destabilize* at time t if the output of v changes to $\perp$ at time t. Finally, both v and its output are said to *transition* at time t if v either stabilizes or destabilizes at time t.

A *clock set* for a level-clocked circuit G is a set containing a clock (clocking waveform) ϕ for each level-clocked latch in G. Our timing analysis algorithms can be applied to any clock set Φ with the following properties:

1. The set Φ is finite, and its elements are fully specified clocks.

2. For any time t, every cycle in G contains at least one latch whose clock has value LOW at time t.

Clock sets with the second property are said to be *fully synchronous*. Our analysis and algorithms are not directly applicable to circuits that "gate" their clock signals and/or rely on "two-sided" timing constraints [6]. In addition, we assume for simplicity that there exists a *start time* $t_0 > -\infty$, such that all clocks in Φ are constant over the interval $[-\infty, t_0]$. Henceforth, we assume these properties hold.

In general, the clocks in a clock set are assumed to repeat after some finite amount of time. A clock set Φ is *periodic*, if there exists a strictly positive real number π, such that $\phi(t) = \phi(t + \pi)$ for all $t \geq t_0$ and $\phi \in \Phi$. The number π is the *period* of Φ.

For any time t and clock set Φ it is possible to divide the interval $[-\infty, t]$ into a finite number of intervals, or *steps*, during which all clocks in Φ hold constant values. Steps are ordered in the natural fashion, and we denote the k^{th} step of the clock set by its index k or by its endpoints $\langle t_k, t_{k+1} \rangle$. (The delimiters "$\langle$" and "$\rangle$" simply indicate that whether the ends of an interval are open or closed depends on context.) By convention, the interval $[-\infty, t_0]$ is the -1^{st} step of the clock set, and t_k always denotes the starting endpoint of the k^{th} step.

Our definition of proper circuit operation is based on a concept of "ideal outputs." We assume that an *ideal circuit* is one whose components have infinitesimal propagation delays. The *ideal output* of a component at time t is the output at time t of the corresponding component in a structurally, and functionally, equivalent ideal circuit. A circuit is said to *operate properly*, if for all time t the outputs of latches whose clocks are LOW at time t are equal to their ideal outputs. This definition of proper operation is similar to the definition of "correct behavior" used by Szymanski [18], and the definition of "intended behavior" used by Weiner and Sangiovanni-Vincentelli [22].

3 Computational Expansions

In this section, we show how to construct circuits that perform in a combinational fashion the same computation as a given circuit G. The construction essentially makes multiple copies of components in G and connects them together in such a way that for every possible transition by some component in G, there exists a copy of the component, in the combinational circuit, which computes the value that the component transitions to. The resulting combinational circuit is a "computational expansion" of G. Our timing analysis algorithms are based on the strong correlations that exist between the operation of G and the operation of a corresponding computational expansion.

Consider the circuit G' shown in Figure 4. The circuit consists of copies of the components from the circuit G in Figure 1. Groups of components in G' are associated with steps of the clock set $\{\phi_1, \phi_2\}$, and we use v_k to denote the copy of component v, in G, that is in the group associated with step k. Latches associated with step -1 have constant LOW clocks, while all other latches have constant HIGH clocks.

The circuit G' performs in a combinational fashion the same computation as the circuit G. If both ϕ_1 and ϕ_2 have value LOW for all time less than 0, and the latches C_{-1} and D_{-1} (in G') are initialized so that they output the values that C and D (in G) hold at time 0, then the ideal output of any component in G over the interval $[-\infty, 1)$ is eventually settled to by a component associated with step -1 in G'. Similarly, the ideal output of any component in G over the interval $[1, 3]$ is eventually settled to by some component associated with steps -1 and 1 in G'. In fact, the ideal output of any component in G for all times less than 37 is eventually settled to by some component in G', *i.e.*, G' computes the ideal outputs of G for all times less than 37. The circuit in Figure 4 is not combinational, in a strict sense, since it includes latches with clock inputs. Like most combinational circuits, however, the circuit is acyclic, and this acyclicity is exploited by our timing analysis.

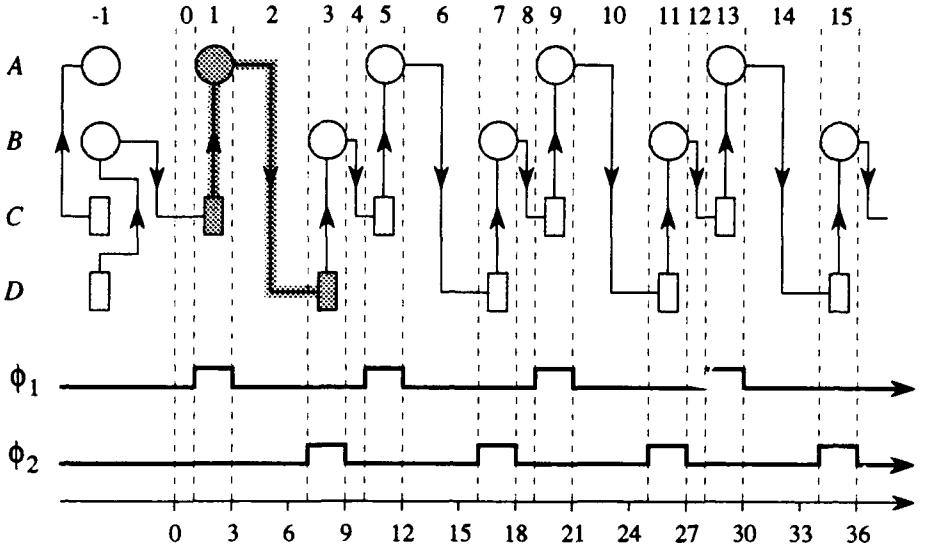

Figure 4: Illustration of computational expansion for the circuit from Figure 1. Dark-shaded path represents a simple delay constraint.

For simple circuits like the one in Figure 1, a computational expansion can be constructed by making a copy of a component for each time that the ideal output of the component changes. Consider again the circuit G' shown in Figure 4. The circuit computes the ideal outputs of the components of G in Figure 1 for steps -1 through 15. In addition, it also computes the ideal outputs for steps -1 through 16, since the fall of ϕ_2 at time 36 cannot cause the ideal output of any component in G to differ between steps 15 and 16. Observe that G' does not compute all the desired ideal outputs for steps -1 through 17, since the rise of ϕ_1 at time 37 (not shown) can cause the ideal outputs of A and C (in G) to differ between steps 16 and 17, thus implying changes in the ideal outputs of A and C which are not "represented" by any component in G'. A computational expansion for steps -1 through 17 can be obtained, however, by augmenting G' with an additional copy A_{17} of A, and an additional copy C_{17} of C. The final output values of A_{17} and C_{17} can be insured to equal the desired ideal output values, by placing edges to the new copies from the "most recent" copies of the components whose outputs are inputs to A and C, *i.e.*, from B_{15} to C_{17}, and from C_{17} to A_{17}. By beginning with the vertex-induced subgraph defined by the components associated with step -1, and inductively repeating the construction just described, a computational expansion of G for steps -1 through n can be constructed for any nonnegative n.

Given some method for establishing the times when the ideal outputs of components change, the construction just described can be easily formalized. Let $I(v,k)$ denote the earliest step such that the ideal output of a component v is constant over the interval $\langle t_{I(v,k)}, t_{k+1}\rangle$, *i.e.*, the most recent step where the ideal output of the component changed. For any component v, the set of steps $\{k : I(v,k) = k\}$ contains exactly one step for each time that the ideal output of v changes value, and thus, a copy v_k of v is needed for each step k such that $I(v,k) = k$. By subscripting copies of components with steps where the ideal outputs of the original components changed, the edges to any copy v_k are easily constructed by noting that the "most recent" copy u_l of any component u whose output is an input to v must be such that $l = I(u,k)$. Consequently, one naive way to construct a "computational expansion" $G_{CX} = (V_{CX}, E_{CX})$ of a given circuit $G = (V,E)$ would be to let

$$\begin{aligned} V_{CX} &= \{v_k : v \in V \text{ and } I(v,k) = k\} \\ E_{CX} &= \{(u_l, v_k) : (u,v) \in E,\ I(u,k) = l, \text{ and } I(v,k) = k\}. \end{aligned}$$

If the clock set is fully synchronous, then G_{CX} is acyclic, except for possibly the subcircuit of copies associated with step -1. Consequently, if every latch v_k such that $k \neq -1$ has a constant HIGH clock, and every latch v_{-1} has a constant clock, whose value is equal to the clock of v at time $-\infty$, and given suitable initialization, a simple inductive argument can be used to show that the ideal output of component v during step i is

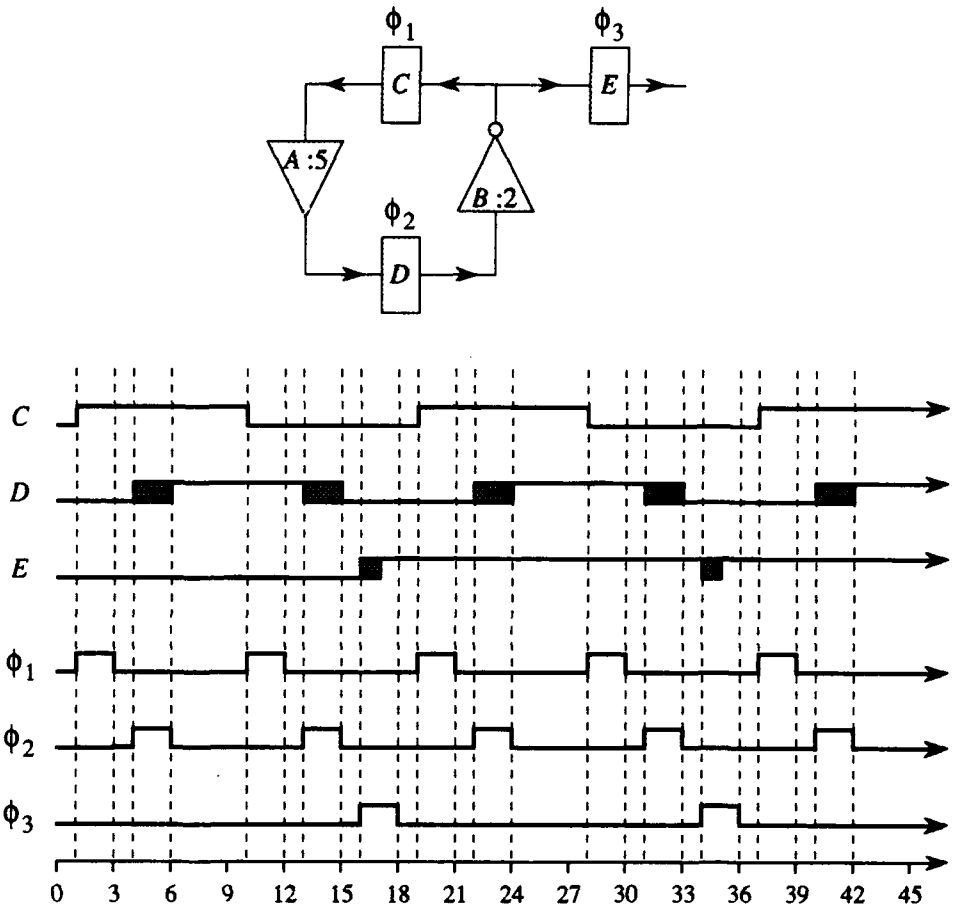

Figure 5: A circuit demonstrating the inability of changes in ideal output to capture all changes in component outputs. Darkly shaded regions represent intervals of time when component outputs are undefined.

eventually output by $v_{I(v,i)}$.

Unfortunately, the construction based on changes in ideal output, does not necessarily generate a copy of a component for each component transition. The problem is that component transitions that occur in the actual circuit may disappear when the circuit is idealized with delays that approach zero. Thus, while the construction based on changes in ideal output would always generate a combinational circuit that computes the same *function* as the original circuit, it would not always generate a combinational circuit which performs in a combinational fashion the same *computation* as the original circuit.

Consider, for example, the simple circuit shown in Figure 5. The functional element A is a simple binary buffer, while the functional element B is a binary inverter. The subcircuit composed of A, B, C, and D essentially forms an unstable inverter ring, and consequently the outputs of all four of these components are expected to flip back-and-forth between logical 1 and 0. Now, due to the large amount of time between the fall of ϕ_2 and the next rise of ϕ_1, the output of B is guaranteed to be stable when ϕ_1 rises, and thus the output of C flips cleanly between 1 and 0, as shown. Even when the output of C flips cleanly, however, the delay of A is sufficiently long to cause A to output an undefined value while the clock to D is HIGH. In fact, the output of A, and subsequently the output of D, does not become defined until the exact moment when ϕ_2 falls. (Fortunately, this is sufficient to insure a properly latched value for our models.) Now, since the output of D does not become stable until the fall of ϕ_2, the output of component B does not become stable until two time units later, and consequently latch E outputs an undefined value, as shown, for one time unit after each rise of ϕ_3. Observe, however, that if the delay of B were less than 1, the output of E would be constant at 1 for all time greater than 16. Clearly, for this circuit, the ideal output of E is constant for all

time greater than or equal to 16, despite the fact that the actual output of E destabilizes after each rise of ϕ_3. Consequently, a construction based on changes in ideal output is not guaranteed to capture every transition that occurs during the computation of a given circuit.

To facilitate the handling of difficulties like the one demonstrated in Figure 5, it is convenient to formalize the notion of a "potential" component transition. Our definition of what constitutes such a transition is based on the behavior of a circuit whose clocks "stop" during the k^{th} step. Formally, given a circuit G and Φ, the k^{th} *approximation* of G is the circuit G^k that is identical to G, but clocked by Φ^k, where the clocks in Φ^k are equal to the clocks in Φ over the interval $[-\infty, t_{k+1})$ but are constant for all time greater than or equal to t_{k+1}, with the values they held during step k. The function $B^*(v,k)$ denotes the earliest step i such that over the interval $[-\infty, \infty]$ the output of vertex v in G^i is equal to the output of v in G^k. The *minimal computational expansion* for a circuit $G = (V, E)$ is the circuit $G^*_{CX} = (V^*_{CX}, E^*_{CX})$ where

$$\begin{aligned} V^*_{CX} &= \{v_k : v \in V \text{ and } B^*(v,k) = k\} \\ E^*_{CX} &= \{(u_l, v_k) : (u,v) \in E, B^*(u,k) = l, \text{ and } B^*(v,k) = k\} \end{aligned}$$

Intuitively, the minimal computational expansion contains a copy of a component v for each step where either the ideal output of v changes or a glitch occurs in the actual output of v, *i.e.*, for each step where the output of v either was *intended* to change or *does* change. Edges insure that each copy of v receives the correct inputs for any particular step. For the sake of clarity, we adopt the convention that components in V are denoted with unsubscripted lowercase letters such as "v" or "u", while *nodes* in V_{CX} are denoted with subscripted lowercase letters such as "v_k" or "u_l". In addition, a node denoted with "v_k" is assumed to be a copy of component $v \in V$ that exists because $B^*(v,k) = k$. Such a node is said to be in the kth *level* of G_{CX}. Every latch v_k such that $k \neq -1$ has a constant HIGH clock. Every latch v_{-1} has a constant clock, whose value is equal to the clock of v at time $-\infty$. Latches whose clocks are LOW at time $-\infty$ are initialized so that they have the same outputs as the corresponding latches in G. The first 19 levels of the minimal computational expansion for the circuit from Figure 5 is shown in Figure 6. The darkly shaded node would not be in the expansion if components were only copied when their ideal output changed.

Unfortunately, it is unlikely that polynomial-time timing verification algorithms can be based on the minimal computational expansion. The difficulty is with the reliance of the definition of B^* on the actual output of a component. Consider, for example, the circuit in Figure 5, and suppose that all we wished to determine was whether $B^*(A,1) = 1$. For the initial conditions shown, $B^*(A,1)$ is obviously 1, since the output of C is initially 0 while the output of component B is initially 1. Observe, however, that if the output of C were instead initially 1, then $B(A,1)$ would be equal to -1. Thus, if we wish to verify the timing of a circuit for all possible initial conditions, many different minimal computational expansions may need to be considered, possibly exponentially many. In addition, for some circuits, particularly those containing "counters," changes in the output of a particular component may manifest themselves only after an exponential number of steps have passed. Unfortunately, both these difficulties are likely to be fundamental, since a reduction from boolean satisfiability [7] shows that timing verification is an NP-hard problem in general.

3.1 Base Step Functions

The key to using computational expansion for efficient timing analysis is to use approximations to G^*_{CX} that are "pessimistic" about when the outputs of components change value. Consider the circuit shown in Figure 7. The circuit is identical to the circuit in Figure 5, except that component A is now an inverter. The subcircuit composed of A, B, C, and D now forms a stable inverter ring, and consequently the minimal computational expansion of the circuit is as shown in Figure 8. Observe, however, that if all the buffers in the expansion from Figure 6 were replaced with inverters, the resulting expansion would, in a sense, approximate the expansion from Figure 8. Specifically, there exists (in the expansion) a copy of a component for each transition of the component. The expansion is only an approximation because the converse is not true, *i.e.*, an actual or intended change in output does not exist for each copy in the expansion. As will be shown in Sections 4 and 6, timing verification based on such approximations will never fail to identify a circuit which does not operate properly. The key to efficient timing verification, is that there exist expansions that are "easy" to generate, yet approximate the minimal computational expansion regardless of what either the initial conditions or functions computed by components might be.

Approximations to the minimal computational expansion are specified using a "base step function." A *base step* of component v at step k is any step i such that the output of vertex v in the approximation G^i is

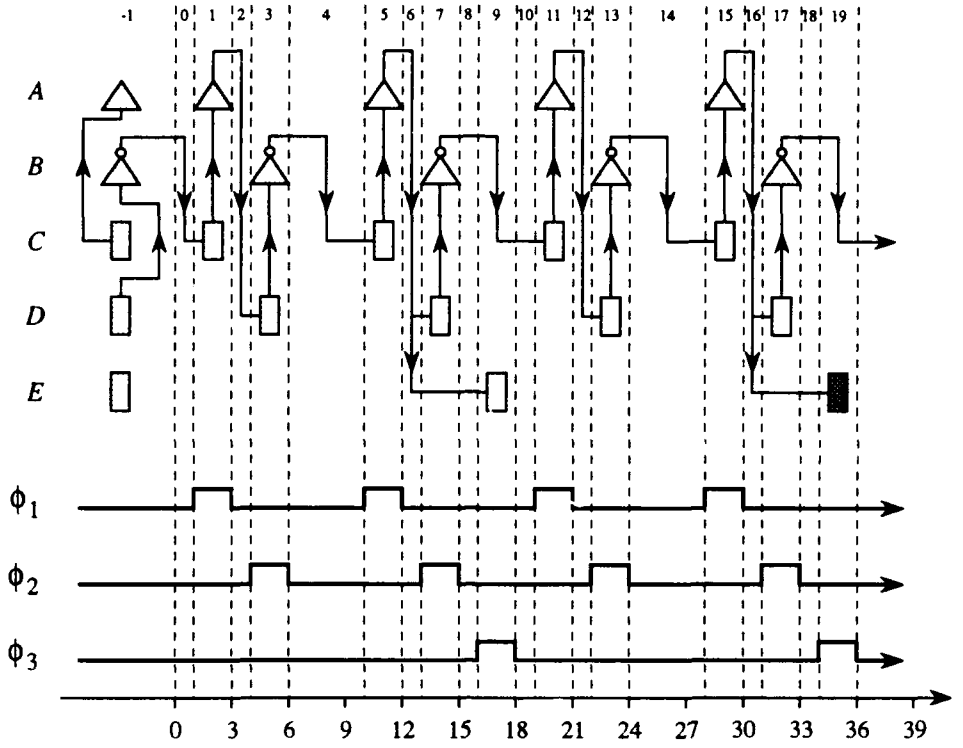

Figure 6: The minimal computational expansion for the circuit from Figure 5. Darkly shaded node would not be present in an expansion generated using changes in ideal output.

equal to the output of v in the approximation G^k over the interval $[-\infty, \infty]$. Given some *base-step function*, B, which maps each component-step pair to a base step, a *computational expansion* of a circuit $G = (V, E)$ is defined to be a graph $G_{CX} = (V_{CX}, E_{CX})$ where

$$\begin{aligned} V_{CX} &= \{v_k : v \in V \text{ and } B(v,k) = k\} \\ E_{CX} &= \{(u_l, v_k) : (u,v) \in E, B(u,k) = l, \text{ and } B(v,k) = k\} \end{aligned}$$

As before, every latch v_k such that $k \neq -1$ has a constant HIGH clock, while every latch v_{-1} has a constant clock, whose value is equal to the clock of v at time $-\infty$. In addition, latches whose clocks are LOW at time $-\infty$ are initialized so that they have the same outputs as the corresponding latches in G. As in the minimal computational expansion, a node denoted with "v_k" is assumed to be a copy of component $v \in V$ that exists because $B(v,k) = k$. Observe, that the definition of G_{CX} differs from the definition of G^*_{CX} only in that copies of components might be made for *any* step that meets the specified conditions, rather than just the *earliest* step.

Intuitively, base step functions are a convenient way to encapsulate the assumptions about "when things change" in a given circuit. The adoption of such assumptions is natural whenever a detailed simulation, like that which may be needed to generate the minimal computational expansion, is considered impractical. Indeed, as noted earlier, a reduction from boolean satisfiability [7] shows that timing verifications is an NP-hard problem in general, and thus, indicates that such assumptions are likely to be necessary if the timings of arbitrary circuits are to be verified within an amount of time which is polynomial in their size. Most of our results are generic in the sense that they can be applied whenever a set of assumptions can be specified as a suitable base step function.

3.2 Expanding Base Step Functions

To apply the timing analysis of Section 4, a computational expansion G_{CX} must have three important properties. First, every cycle in G_{CX} must be broken by some latch whose clock is LOW. Second, if step i is

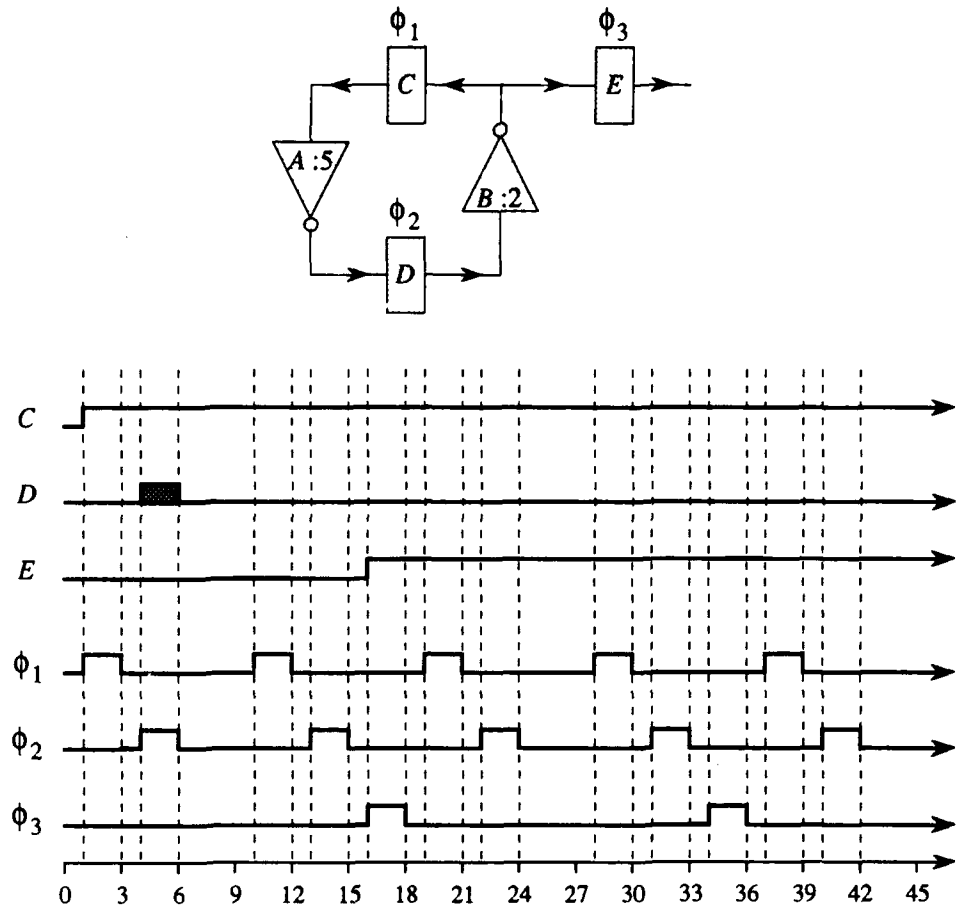

Figure 7: A circuit whose minimal computational expansion can be approximated by replacing buffers with inverters in the computational expansion shown in Figure 6. The minimal computational expansion of the circuit is shown in Figure 8.

the base step specified for the component-step pair (v, k), then the node v_i must compute the ideal output of component v in G during step k of Φ. Third, for every edge (v_k, u_l) in G_{CX}, we have $k \leq l$, *i.e.*, edges never go from one level of the expansion to a lower level of the expansion. Any base step function that is guaranteed to generate computational expansions with these properties is said to be an *expanding* base step function. Most base step functions of interest can be shown to be expanding base step functions, using arguments similar to those used to prove the following lemma.

Lemma 3.1 *For any circuit $G = (V, E)$ and fully synchronous clock set Φ, there exists an expanding base step function.*

Proof: (sketch) Proof of the lemma is by construction, and makes use of two orderings based on the clocks in Φ. The first is a derived partial ordering on the components in G, while the second is the natural total ordering that exists on the steps of Φ.

A base step function B which satisfies the lemma can be defined as follows. For any latch v whose clock is LOW during step k, let step i be the least step such that the clock of v is LOW during all steps greater than i and less than or equal to k, *i.e.*, the clock of v is LOW over the entire interval $\langle t_{i+1}, t_{k+1} \rangle$. The function B maps (v, k) to step i when v is a latch whose clock is LOW during step k, and maps (v, k) to step k otherwise. The computational expansion that results from B contains no cycles that are unbroken by a latch whose clock is LOW, since the synchronous nature of G prevents any such cycles within the -1^{st} level of the computational expansion, and guarantees that the rest of the computational expansion is in fact acyclic.

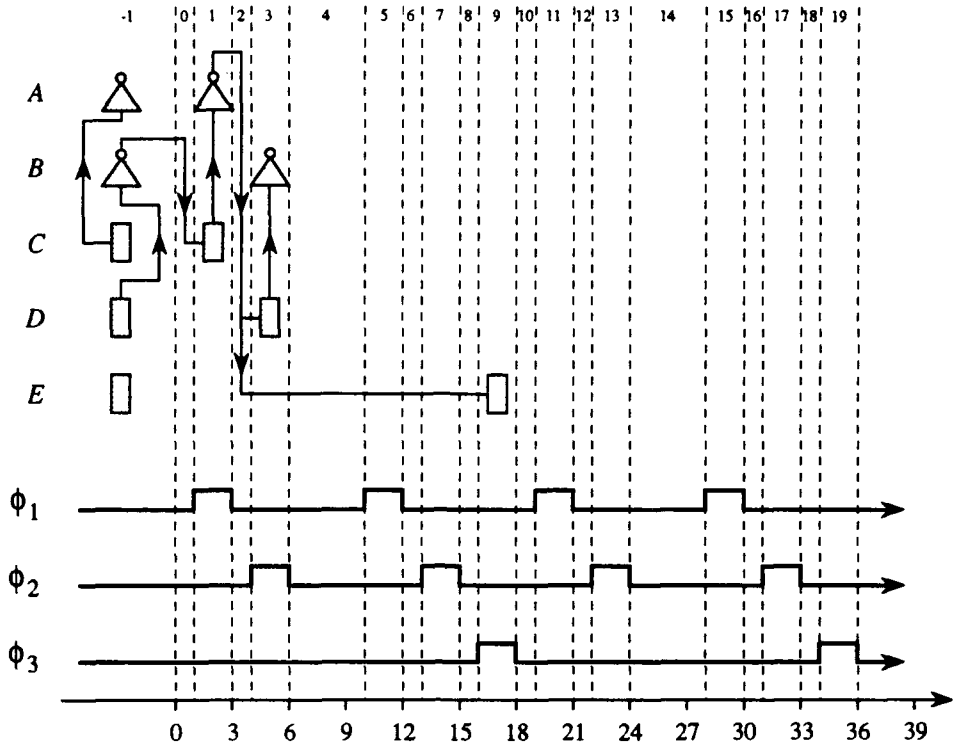

Figure 8: The minimal computational expansion for the circuit from Figure 7. The expansion can be approximated by replacing buffers with inverters in the computational expansion shown in Figure 6.

To define the derived partial ordering that is used to prove that B is a base-step function, consider the circuit $G_k = (V, E_k)$, where $(v, u) \in E_k$ if and only if $(v, u) \in E$ and u is not a latch whose clock input has value LOW during step k. Since Φ is fully synchronous, the circuit G_k must be acyclic, and thus the edges in G_k define a partial order on the components in V. The defined partial order is the kth *configuration order*, where a component v is before a component u if and only if there exists a path from v to u in G_k. If we assume that all functional elements have at least one input,[1] then latches whose clocks are LOW during step k are the only components which do not have some other component before them in the kth configuration order. In addition, since G_k is acyclic, if any component in V has a given property, then there must exist some component with the property, that is *first* in the sense that no other components with the property are before it in the kth configuration order. By showing that no such first component can exist, the kth configuration order can be used to establish that no component has a given property.

An inductive argument can be used to show that B computes base steps. First, we hypothesize inductively that B computes base steps for all steps less than or equal to k and then use the $(k+1)$st configuration order to show that B computes base steps for step $k+1$. Since the hypothesis is obviously true for $k = -1$, the function B must, by induction, compute base steps for all component-step pairs.

The proof that B is an expanding base step function is analogous to the proof that B computes base steps. By using configuration orders to induct on steps, each of the three properties of expanding base step functions can be shown for B. ∎

It is important to realize that the models from Section 2 are assumed throughout this paper. This assumption implies many "natural" properties which are not stated explicitly. For example, Equations 1 and 2 guarantee that the outputs of components change only in response to a change in some input, and *must* become constant after some appropriate delay. Without properties such as these, the proof of Lemma 3.1 would not be valid, and the timing constraints to be presented in Section 4 would be of limited use.

[1] Given the form of Equation 1, any functional element with no inputs would have a constant output value. Consequently, such a functional element could be deleted from the circuit, as long as components that used the output of the deleted functional element were suitably modified.

$$
\begin{array}{lcl}
9-1 & \geq & 5 \\
12-1 & \geq & 5+4 \\
12-7 & \geq & 4 \\
18-1 & \geq & 2\cdot 5+4 \\
18-7 & \geq & 5+4 \\
18-10 & \geq & 5
\end{array}
\qquad
\begin{array}{lcl}
21-1 & \geq & 2\cdot 5+2\cdot 4 \\
21-7 & \geq & 5+2\cdot 4 \\
21-10 & \geq & 5+4 \\
21-16 & \geq & 4 \\
27-1 & \geq & 3\cdot 5+2\cdot 4 \\
27-7 & \geq & 2\cdot 5+2\cdot 4 \\
27-10 & \geq & 2\cdot 5+4 \\
27-16 & \geq & 5+4 \\
27-19 & \geq & 5
\end{array}
$$

Figure 9: Illustration of delay constraints for the circuit from Figure 1.

4 Timing Constraints

In this section, we define a set of constraints which can serve as a set of sufficient conditions for proper circuit operation. Despite its infinite size, the set of constraint equations is important, since it can be used to guarantee proper circuit operation, handling even unconventional circuits. In Sections 5 and 6, methods are presented that can check the infinite number of constraints quickly.

A computational expansion of a circuit provides a framework for examining the delay constraints described in Section 1. Consider again the computational expansion shown in Figure 4. Each node in the computational expansion corresponds to an output value change in the circuit in Figure 1. For example, C_1 exists in the computational expansion, because a new value propagates to the output of C when ϕ_1 changes value to HIGH at time t_1. This change in the output of C implies subsequent changes in the outputs of A and D, and these changes are reflected by the existence of A_1 and D_3 in the computational expansion. The output of D_3 eventually settles to the ideal output of A, over the interval $[t_1, t_5)$, that must be latched by D at time t_4. Consequently, the delay d_A of A must be less than or equal to the difference between t_4 and t_1 if D is to hold its ideal output over the interval (t_4, t_7). This delay constraint of $t_4 - t_1 \geq d_A$ is represented by the dark-shaded path shown in Figure 4. Using reasoning similar to the above, all the constraints listed in Figure 9 (and many others) can be obtained.

The delay constraints for a circuit $G = (V, E)$ can be specified formally, using an expanding base step function B and the computational expansion $G_{CX} = (V_{CX}, E_{CX})$ generated using B. Let $v \in V$ be any latch whose clock changes value from HIGH during step $k-1$ to LOW during step k. If $B(v, k-1) = i$, then the ideal output of v over the interval $\langle t_k, t_{k+1} \rangle$ is the value computed by $v_i \in V_{CX}$. Thus, at time t_k, v must latch the value computed by v_i. We indicate this fact by associating with v_i a *down-time* of t_k. In a symmetric fashion, let u be any latch where $B(u, j) = j$. Either the ideal output of u changes at time t_j to the value computed by u_j, or the output of u may experience some type of temporary "glitch." We indicate this by associating with u_j an *up-time* of t_j. The set $\Delta(G, B)$ of timing constraints is defined as follows:

$$
\Delta(G,B) = \{t_k - t_j \geq d(\sigma) : \begin{array}{l} v_i \text{ has down-time } t_k,\ u_j \text{ has up-time } t_j, \text{ and} \\ \sigma \text{ is a path in } G_{CX} \text{ from } u_j \text{ to } v_i\}, \end{array}
$$

where $d(\sigma)$ equals the total propagation delay of all nodes in the path σ. By convention, a path from u_j to v_i includes the nodes u_j and v_i, and thus we on occasion use $d(v_i)$ to denote the propagation delay of a single node v_i. Observe that for periodic clock sets, the infinite size of G_{CX} implies that $\Delta(G, B)$ contains an infinite number of constraints. We call each constraint in $\Delta(G, B)$ a Δ-*constraint*.

Although the constraint set $\Delta(G, B)$ is problematic due to its infinite size, it is the only set of constraints that we need to consider. The "scheduling" constraints mentioned in Section 1 can be ignored, since they do not imply any constraints that are not included in $\Delta(G, B)$. The following theorem confirms this fact.

Theorem 4.1 *If all the constraints in $\Delta(G, B)$ are met, then G operates properly.*

Proof: The proof has two parts. The first part shows that if the constraints in $\Delta(G, B)$ are met, then replacing the constant clocks of G_{CX} with a simple clock set based on Φ does not change the final outputs of any nodes in G_{CX}. Since the base step function used to generate G_{CX} is assumed to be expanding, this

is equivalent to showing that latches in G_{CX} latch the appropriate ideal outputs of latches in G, when G_{CX} is clocked with the new clock set. The second part of the proof shows that at any step k it is possible to replace the the first k levels of G_{CX} with the approximation G^k without altering the values latched by the remaining nodes in G_{CX}. This is sufficient to prove the theorem, since if latches in G did not hold their correct ideal outputs at step k, then G^k would input the wrong values into the remaining nodes of G_{CX}, and the replacement would not be possible.

PART 1: Let G_{CX} be clocked by a new clock set Φ_{Simple} that associates with each latch v_k the clock ϕ_{v_k}, where ϕ_{v_k} is defined as follows: if v_k has up-time t_k but no down-time, then ϕ_{v_k} is LOW for all time less than t_k and HIGH for all time greater than or equal to t_k. Similarly, if v_k has down-time v_l but no up-time, then ϕ_{v_k} is HIGH for all time less than or equal to t_l and LOW for all time greater than t_l. Finally, if v_k has up-time t_k and down-time t_l, then ϕ_{v_k} is HIGH during the interval $[t_k, t_l]$ and LOW otherwise. All other nodes use clocks identical to those in the original clock set of G_{CX}. Nodes whose clocks were previously HIGH for all time, but whose clocks in Φ_{Simple} initially have value LOW, are initialized to output $\perp$. We essentially need to show that all latches in G_{CX} latch the appropriate ideal outputs of latches in G.

If some latch in G_{CX} does not latch the appropriate ideal output, then it is possible to identify a set of node-time pairs that are, in some sense, responsible for the failure. Let v_k be a level-clocked latch with down-time t_l. If the output of v_k at time t_l is not the appropriate ideal output, then the input to v_k must not have been the appropriate ideal output at time t_l. Similarly, if the input to v_k is the output of some functional element u_j, then the output of some input to u_j must not have been the appropriate ideal output at time t_l *minus* the propagation delay $d(u_j)$ of u_j. If w_i is the node in question, whose output is the input of u_j, then the node-time pairs (v_k, t_l), (u_j, t_l) and $(w_i, t_l - d(u_j))$ are *late pairs* that prevent v_k from latching the appropriate ideal output. Continuing in this fashion, we can identify the set of all late pairs for v_k. A late pair (w_i, t_k) is *before* another late pair (u_j, t_l) if G_{CX} contains a path from w_i to u_j.

Now, if there exist latches in G_{CX} that do not latch the appropriate ideal outputs, then we can identify at least one late pair that is an "unprovoked" late pair. Let v_k be any latch that does not latch the appropriate ideal output and whose set of late pairs does not contain any other latches that do not latch their appropriate ideal outputs. Since all cycles in G_{CX} must be broken by at least one latch whose clock is always LOW, and G_{CX} has only a finite number of nodes in its first l levels, for any $l \geq -1$, an induction on the structure of G_{CX} can be used to show that such a v_k must exist. Similarly, an inductive argument can also be used to show that the set of late pairs for v_k must contain at least one *unprovoked* late pair that has no late pairs before it.

If, however, the constraints in $\Delta(G, B)$ are met, then no unprovoked late pair can exist, as we now show. Assume that there exists an unprovoked late pair (u_j, t) for v_k, where v_k has down-time t_l, $v_k$does not latch the appropriate ideal output, and the set of late pairs for v_k does not contain any other latches that do not latch the appropriate ideal outputs. Node u_j must be either a functional element, a latch whose clock signal is LOW for all time, or a latch whose clock signal is HIGH over some interval of time. Now, u_j cannot be a functional element, since some node whose output is an input to u_j would be part of a late pair that was before (u_j, t). Also, u_j cannot be a latch whose clock signal is LOW for all time, since this would imply that G_{CX} was not properly initialized. Finally, the fact that all constraints in $\Delta(G, B)$ are met, implies that u_j cannot be a latch whose clock is HIGH over some interval of time. Consider the most complex case of the clock signal being HIGH over some closed interval $[t_m, t_n]$ where $t_m \neq -\infty$. The time t cannot be greater than t_n, since this would violate our definition of v_k by implying that u_j did not latch the appropriate ideal output. In addition, t cannot be less than t_m, since this would imply that $t_l - t_m \leq d(\sigma)$, for some path σ from u_j to v_k, and thus that some $\Delta(G, B)$ constraint was violated. Consequently, the clock of u_j must be HIGH at time t, and thus u_j cannot be a latch whose clock is HIGH over some interval of time, since the component whose output is the input to u_j would be part of a late pair that was before (u_j, t).

Thus, since no unprovoked late pair can exist when all constraint in $\Delta(G, B)$ are met, all latches in G_{CX} clocked with Φ_{Simple} must latch the appropriate ideal outputs whenever all constraints in $\Delta(G, B)$ are met.

PART 2: This part of the proof makes direct use of the k^{th} approximation of G. The outputs of components in G^k are certainly equal to the outputs of components in G over the interval $[-\infty, t_{k+1})$, and thus, the final values latched by latches in G^k are equal to the values latched during step k by the corresponding latches in G. We need to show that for any $k \geq -1$, we can replace the nodes in levels -1 through k with G^k, without

affecting the values latched by the remaining nodes of G_{CX}.

If a component v_{-1} in level -1 of G_{CX} were removed and all edged from v_{-1} were replaced with edges from the copy of v in G^{-1}, then all remaining components in G_{CX} would still operate as before. To see this, simply observe that G^{-1} and level -1 of G_{CX} are essentially identical, since they are are clocked by identical clock sets.

We now complete the proof by showing that if for all $-1 \leq i \leq k$, the nodes in levels -1 through i of G_{CX} can be replaced with G^i, then the nodes in levels -1 through $(k+1)$ of G_{CX} can be replaced with G^{k+1}. We use two steps to show that the replacement works for level $(k+1)$. First, we show that replacing the first k levels of G_{CX} with G^{k+1} also does not affect whether the remaining nodes in G_{CX} latch the appropriate ideal outputs. Second, we show that edges from nodes in level $(k+1)$ of the G_{CX} can then be replaced with edges from the components in G^{k+1} while still not affecting the values latched by the remaining nodes in G_{CX}.

We can replace G^k with G^{k+1} if the output of component v in G^k is equal to the output of v in G^{k+1} whenever $B(v, k+1) \neq k+1$. By the definition of "base-step," however, the output of v in G^k must be equal to the output of v in G^{k+1} whenever $B(v, k+1) \neq k+1$. The outputs of components v such that $B(v, k+1) = k+1$ are of no consequence, since v_{k+1} exists in G_{CX}, and thus there are no edges between v in G^k and the rest of G_{CX}.

To show that edges from nodes in level $(k+1)$ of G_{CX} can be replaced with edges from components in G^{k+1}, we use an argument similar to that used in Lemma 3.1. We need to show that there can be no component v that is the first in the $(k+1)$st configuration order to be such that the edges from node v_{k+1} cannot be replaced with edges from component v in G^{k+1}. No functional element can be such a first component, since the outputs of v and v_{k+1} are identical. To see this, simply observe that v being first in the $(k+1)$st configuration order implies that all edges to v_{k+1} can be replaced with edges from components in G^{k+1}. A more intricate argument also shows that a latch whose clock is HIGH during step $k+1$ cannot be such a first component. If the clock of v is HIGH for all time, then the clock of v_{k+1} must also be HIGH for all time, and the outputs of v and v_{k+1} must once again be identical. If the clock of v is not HIGH for all time, then let t_i be the time that the clock last changed value from LOW to HIGH. Before time t_i, the output of v_{k+1} must be $\perp$ and consequently the output of v_{k+1} is of no consequence before t_i. Thus, since the outputs of v and v_{k+1} are certainly identical for all time greater than or equal to t_i, we can replace edges from v_{k+1} with edges from v, without affecting the remaining nodes in G_{CX}. Finally, by combining the argument for latches whose clocks are HIGH during step $k+1$ with the fact that a latch v_{k+1} latches the appropriate ideal output when nodes in earlier levels are replaced by G^k, it can be shown that a latch whose clock is LOW during step $k+1$ also cannot be such a first component. ∎

Now that the set of Δ-constraints has been defined, it is possible to appreciate why certain "natural" functions are not sufficiently general for our purposes. For example, the function I would initially seem to be a reasonable basis for the construction of a computational expansion, but as illustrated in Figure 6, there exist circuits for which the function I can result in expansions which are missing potentially important nodes. Without these nodes, the set of Δ-constraints define by the expansion would be incomplete.

Similarly, functions that only consider changes in the value output by a component are also not sufficiently general to handle all circuits. For example, consider the function B^S that maps a component-step pair (v, k) to the earliest step i such that during the interval $\langle t_i, t_{k+1} \rangle$ the ideal output of v is constant, and v never destabilizes, *i.e.*, the earliest step i such that the output of v is "settling" over the interval $\langle t_i, t_{k+1} \rangle$. While for many level-clocked circuits, B^S is likely to be a base-step function, there do exist circuits for which B^S is not sufficiently general. To see this, consider the circuit fragment shown in Figure 10. The functional element A is a binary OR-gate whose propagation delay is 4. The first thing to note about the fragment is that the destabilization of A at time 1 does not correspond to a change in ideal output. It is thus apparent that the shown waveforms reflect an assumption that the output of A "glitches" each time that one of the inputs to A changes value. Since many implementations of an OR-gate do not have this property, the example is not particularly realistic. Nevertheless, the fragment serves its intended illustrative purpose. A consequence of the "glitch" property, is that the output of A does not stabilize until one propagation delay after the rising edge of ϕ_2 at time 13, and thus, the output of A in invalid over the entire interval $[10, 17]$. Observe, that this interval is 3 time units longer than the propagation delay of A. The extreme length of the interval is due to the fact that two "invalid" intervals of length 4 are overlapping. Unfortunately, B^S does not yield an expansion which reflects the length of the interval. If one applies the various definitions, one finds that the minimal computational expansion of the fragment is the "circuit" shown in Figure 11. An expansion based

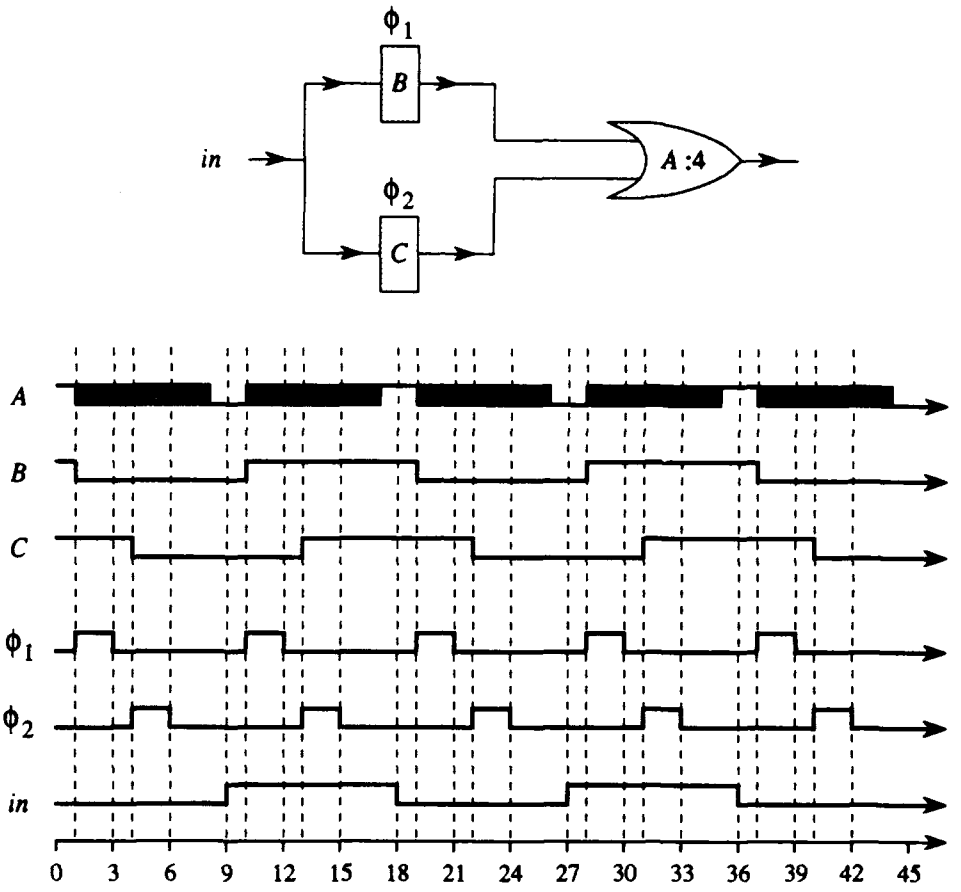

Figure 10: A circuit demonstrating the inability of "settling" to capture all computations by component. The minimal computational expansion of the circuit is shown in Figure 11.

on B^S, however, would not include the darkly shaded nodes, since, for example, the output of A is settling over the interval $[10, 19]$, and the ideal output of A is constant over that interval, as well.

Given the amount of "mechanism" needed to define the set of Δ-constraints, it is natural to ask whether the complexity is justified. There are two primary benefits of defining the set of Δ-constraints in terms of a computational expansion and a base-step function. First, the base-step function provides a mechanism for tailoring the set of Δ-constraints to the peculiarities of a particular circuit. For example, one could define a base-step function which reflected the behavior of stable feedback loops, or multiplexors. Second, the definition of what constitutes a base-step function provides a precise criterion for what properties the user is guaranteeing when he specifies a "customized" base-step function. Moreover, the following theorem essentially states that the required properties are precisely the ones that are needed for accurate timing verification.

Theorem 4.2 *For any externally synchronized circuit G, if some constraint in $\Delta(G, B^*)$ is not satisfied, then G is not properly timed.*

Proof: (sketch) Proof of the theorem follows the same basic outline as the proof of Theorem 4.1. Here, however, the existence of a violated Δ-constraint immediately implies that some latch in G_{CX} must latch $\perp$ if G_{CX} is clocked by Φ_{Simple}. The second step of the proof shows that if some latch in G_{CX} latches $\perp$, then some latch in G must also latch $\perp$, and thus that G is not properly timed.

An argument similar to one used to prove Lemma 3.1 can used to prove that if some latch in G_{CX} latches

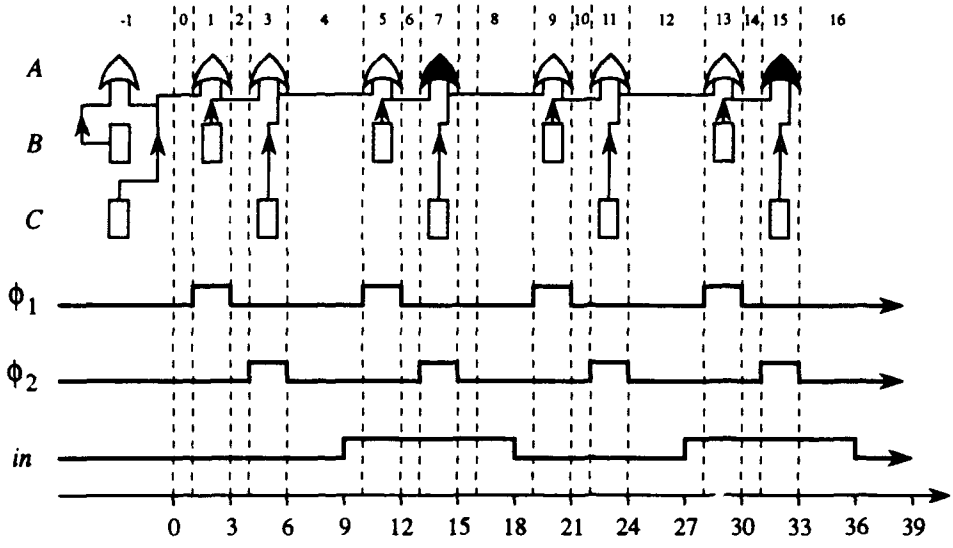

Figure 11: The minimal computational expansion of the circuit shown in Figure 10. The darkly shaded nodes would not be present in a expansion based on B^S.

$\perp$, then some latch in G must also latch $\perp$. Let $\mathcal{P}$ be the following predicate:

$$\mathcal{P}(v,t) = \begin{cases} \text{TRUE} & \text{if there exist } t' \leq t \text{ such that, at time } t', \\ & \text{the output of } v \text{ is not equal to the output of} \\ & v_i, \text{ where } i \text{ is the base step for } v \text{ and} \\ & \text{the step containing } t' \\ \text{FALSE} & \text{otherwise.} \end{cases}$$

If $\mathcal{P}$ is FALSE for all vertex-time pairs, then the output of any vertex v during step i must be equal to the output of node $v_{B^*(v,i)}$, in the computational expansion, over the interval $\langle t_i, t_{i+1} \rangle$. Thus, if $\mathcal{P}$ is always FALSE, then some latch in the original circuit must latch $\perp$, whenever some latch the minimal computational expansion latches $\perp$. The proof that $\mathcal{P}$ is FALSE for all vertex-time pairs parallels the argument used in Lemma 3.1. As in the proof of Lemma 3.1, the goal is to use the natural ordering of steps and the configuration orderings of vertices to identify the "first" vertex-time pair for which $\mathcal{P}$ is FALSE. A case analysis can be used to show that no such vertex-time pair can exist. ∎

5 Practical Timing Analysis

In this section, we begin the process of adapting the constraint set Δ for use in timing verification algorithms. First, we examine the implications of different possible base step functions, and present a base step function $\widehat{B}$ which yields constraints that are less pessimistic than the constraints used by previous timing verification algorithms [1, 4, 8, 9, 11, 14, 15, 16, 17, 21, 22]. Second, we show how to eliminate redundant constraints in Δ and obtain a new constraint set δ whose size grows linearly with the size of the computational expansion. The definitions of $\widehat{B}$ and δ immediately yield a simple algorithm for verifying the proper operation of any circuit that computes for only a finite number of steps. The algorithm is quite general, and is applicable to circuits using nonperiodic clock sets. In addition, some of these results are used in Section 6, where we describe an algorithm for verifying the proper operation of circuits that compute indefinitely with periodic clock sets.

5.1 Base Step Functions

The base step function greatly affects the usefulness of the constraint set Δ. Ideally, the constraints in Δ would be a necessary and sufficient set of conditions for the proper operation of a circuit. Unfortunately,

whereas any base step function yields constraints that are sufficient for proper circuit operation, most base step functions yield constraints that are not necessary; a circuit may operate properly even if it violates some of the constraints. In general, the constraints in Δ are not a necessary set of conditions unless the base step function is essentially equal to B^*. Consequently, computing a necessary and sufficient set of conditions is unlikely to be computationally tractable, since a simple reduction from boolean satisfiability [7] shows that the problem of computing B^* is NP-hard.

In some cases, it is possible to order different base step functions according to how closely the set of constraints that they yield approximates a set of necessary conditions. Specifically, a base step function B is *less strict* than another base step function B', if all circuits that meet the Δ-constraints yielded by B' also meet the Δ-constraints yielded by B, but some circuits that meet the Δ-constraints yielded by B may not meet the Δ-constraints yielded by B'. Intuitively, B is less strict than B', if it disqualifies fewer properly operating circuits.

The delay constraint equations used by previous timing verifiers loosely corresponds to the Δ-constraints yielded by the following recursive base step function:

$$B_{\text{trad}}(v,k) = \begin{cases} \max_{(u,v)\in E} B_{\text{trad}}(u,k) & \text{if } k \neq -1, \text{ and } v \text{ is a functional element} \\ B_{\text{trad}}(v,k-1) & \text{if } k \neq -1, \text{ and } v \text{ is a latch whose clock is LOW during step } k \\ k & \text{if } k \neq -1, \text{ and } v \text{ is a latch whose clock is LOW during step } k-1 \text{ and HIGH during step } k \\ \max(B_{\text{trad}}(v,k-1), B_{\text{trad}}(u,k)) & \text{if } k \neq -1, (u,v) \in E, \text{ and } v \text{ is a latch whose clock is HIGH during steps } k-1 \text{ and } k \\ -1 & \text{if } k = -1 \end{cases}$$

Ignoring the last "initialization" case, the function B_{trad} essentially states that functional elements find their base steps by taking the maximum over the base steps of all components that are inputs to them. Latches whose clocks are LOW have a constant base step, while latches whose clocks change from LOW to HIGH change base step to the step after the clock transition occurs, and behave like functional elements as long as their clocks remain HIGH. Unfortunately, the function B_{trad} always disqualifies the circuit shown in Figure 2, because of the apparent delay constraint violation mentioned in Section 1.

A more sophisticated base step function results in a set of Δ-constraints which disqualifies fewer properly operating circuits than previous timing verification algorithms. Previous algorithms essentially assume that the output of a latch changes whenever its clock changes from LOW to HIGH. This assumption is unnecessary, since the base step of the input to the latch provides a much more reasonable indication of whether a change in the output has occurred. One base step function $\widehat{B}$ which incorporates this idea is recursively defined as follows:

$$\widehat{B}(v,k) = \begin{cases} \max_{(u,v)\in E} \widehat{B}(u,k) & \text{if } k \neq -1, \text{ and } v \text{ is a functional element} \\ \widehat{B}(v,k-1) & \text{if } k \neq -1, \text{ and } v \text{ is a latch whose clock is LOW during step } k \\ \widehat{B}(v,k-1) & \text{if } k \neq -1, (u,v) \in E, v \text{ is a latch whose clock is HIGH during step } k, \text{ and } \widehat{B}(v,k-1) \geq \widehat{B}(u,k) \\ k & \text{if } k \neq -1, (u,v) \in E, v \text{ is a latch whose clock is HIGH during step } k, \text{ and } \widehat{B}(v,k-1) < \widehat{B}(u,k) \\ k & \text{if } k \neq -1, \text{ and } v \text{ is a latch whose clock is HIGH during step } k \text{ and LOW during steps } -1 \text{ through } k-1 \\ -1 & \text{if } k = -1 \end{cases}$$

The base step function $\widehat{B}$ differs from B_{trad} in the way it handles a latch whose clock changes value from

LOW to HIGH. Rather than assume a change in the base step of a latch whose clock changes value from LOW to HIGH, $\hat{B}$ considers whether the base step of the input has changed since the step when the clock last changed from HIGH to LOW. If the base step of the input has not changed, then the value that passes through the latch when the clock becomes HIGH will be the same value that was latched when the clock last became LOW. In such a case, no change to the base step of the latch is necessary. Inductive arguments can be used to show that the function $\hat{B}$ is an expanding base step function. Unlike B_{trad}, $\hat{B}$ does not disqualify the circuit in Figure 2.

The fifth clause in the definition of $\hat{B}$ reflects an assumption that the output of a latch whose clock is initially LOW always changes value the first time that the clock for the latch becomes HIGH. This assumption is by no means arbitrary, since it greatly simplifies the verification of circuits that use periodic clock sets. In Section 6, we will discuss the implications of this assumption, and possibilities for how it can be removed.

Not surprisingly, it is possible to show that $\hat{B}$ is less strict than B_{trad}. The proof makes use of the fact that $i \geq j$ implies that $\hat{B}(v,i) \geq \hat{B}(v,j)$ and $B_{\text{trad}}(v,i) \geq B_{\text{trad}}(v,j)$, *i.e.*, both base step functions are *monotone*. In fact, it is possible to show the following general lemma:

Lemma 5.1 *If B and B' are two monotone base step functions, and $B(v,k) \leq B'(v,k)$ for all components v and steps k, then all circuits that meet the Δ-constraints yielded by B' also meet the Δ-constraints yielded by B.*

Proof: We show that any circuit that does not meet the Δ-constraints yielded by B also does not meet the Δ-constraints yielded by B'. Let $t_k - t_j \geq d(\sigma)$ be a violated Δ-constraint yielded by B, where the path σ is from u_j to v_i, and v_i has down-time t_k and u_j has up-time t_j. We show that the computational expansion yielded by B' contains a path σ' from some u_n to some v_m, such that v_m has down-time t_k, u_n has up-time t_n greater than or equal to t_j and the total delay along σ' is equal to the total delay along σ. Such a σ' directly implies the violation of a Δ-constraint yielded by B'.

We demonstrate the existence of σ' with an explicit construction. Each node in σ has a corresponding node in σ'. The construction begins with a node that corresponds to v_i in σ and inductively proceeds backwards along a path that eventually becomes σ'. Successive pairs of corresponding components maintain the invariant that the node in σ is never in a higher level than the corresponding node in σ'.

To find the node v_m that corresponds to v_i, simply note that v_i has a down-time of t_k only if $B(v,k) = i$. Thus, we let v_m be such that $B'(v,k) = m$. We know that $m \geq i$, since the result of B' is greater than or equal to the result of B for any component-step pair.

Given any corresponding pair of nodes in σ and σ', the next corresponding pair of nodes is found by simply tracing back through the respective computational expansions. Let w_p and w_q be a corresponding pair of nodes in σ and σ', respectively. If the component before w_p in σ is x_r, then the component before w_q in σ' is x_s, where $B'(x,q) = s$. The conditions of the lemma specify that $B'(x,q) \geq B(x,q)$. In addition, since the invariant between corresponding pairs of nodes guarantees that $q \geq p$, the monotonicity of B implies that $B(x,q) \geq B(x,p)$. Consequently, since $B(x,p) = r$, we can conclude that $s \geq r$, and thus that x_r and x_s maintain the invariant between pairs of nodes.

The final σ' can be used to identify a violated Δ-constraint yielded by B'. By construction, the total delay along σ' must be equal to the total delay along σ. In addition, since $B'(v,k) = m$, the last node v_m in σ' must have a down-time of t_k. Finally, the invariant between corresponding nodes in σ and σ' guarantees that the first node u_n in σ' cannot be in a lower level of the computational expansion than the first node u_j of σ, and thus, that u_n must have a up-time t_n that is greater than or equal to the up-time of u_j. Consequently, the fact that the Δ-constraint associated with σ is violated, directly implies that the Δ-constraint associated with σ' must also be violated. ∎

Corollary 5.1 *$\hat{B}$ is less strict than B_{trad}.*

Proof: As noted earlier, B_{trad} disqualifies the circuit shown in Figure 2, while $\hat{B}$ does not. In addition, one can shown that both $\hat{B}$ and B_{trad} are monotone, and thus the corollary follows immediately from Lemma 5.1. ∎

While not necessary for all of the results in this paper, monotonicity is a natural property for a base step function to have, since any base step function B which is not monotone can easily be transformed into a monotone base step function B' by simply letting $B'(v,j) = B(v,i)$ whenever $B(v,i) < B(v,j)$ for $i > j$. A simple check of the definition of base step shows that the function B' is still a base step function, since the

fact that $B(v,i) < B(v,j)$ directly implies that $B(v,i)$ is also a base step for the pair (v,j). Computational expansions generated with monotone base step functions are *monotone* computational expansions.

5.2 Removing redundant constraints

Careful examination suggests that the majority of the constraints in Δ are redundant. Consider, for example, the Δ-constraints for the computational expansion shown in Figure 4. The down-time 18 that is associated with D_7 is part of three Δ-constraints: $18-1 \geq 2\cdot 5+4$, $18-7 \geq 5+4$ and $18-10 \geq 5$. Observe, however, that if we rewrite the constraints as $18 \geq 2\cdot 5+4+1$, $18 \geq 5+4+7$ and $18 \geq 5+10$, it is apparent that if the down-time associated with D_7 is large enough to satisfy the second of the three constraints, then the down-time satisfies the other two constraints as well. Thus, the first and last of the three constraints are redundant.

In order to avoid such redundancies, we formulate a derived constraint set δ. Consider the set of all constraints in Δ that correspond to paths ending at a particular node v_k in the computational expansion. Each constraint is of the form $t_l - t_i \geq d(\sigma)$, where t_l is the down-time associated with v_k and t_i is the up-time associated with the first node in the path σ. If a particular constraint is such that the quantity $t_i + d(\sigma)$ is maximal, then that constraint is defined to be a *δ-constraint* of v_k. The constraint set δ contains one δ-constraint for each node in the computational expansion. Certainly, all constraints in Δ are met if and only if all constraints in δ are met.

The attractive feature of δ is that the constraints that it contains are easy to generate, and check, whenever a monotone base step function is used. The key observation is that the maximal quantity $t_i + d(\sigma)$ exists for all nodes in the computational expansion, not just latches with associated down-times. Let v_k be any node in a computational expansion generated with a monotone base step function. If σ is a path to v_k from a latch with an associated up-time t_i such that the quantity $t_i + d(\sigma)$ is maximal, then the quantity $db(v_k) = t_i + d(\sigma)$ is the *down-time bound* of v_k. The down-time bound $db(v_k)$ is simple to calculate, using the following recursive definition:

$$db(v_k) = \begin{cases} d(v_k) + \max_{(u_l,v_k)\in E_{CX}} db(u_l)) & \text{if } v_k \text{ is a functional element,} \\ \max(db(u_l), t_k) & \text{if } v_k \text{ is latch with associated up-time } t_k \text{ and } (u_l, v_k) \in E_{CX}, \\ -\infty & \text{if } k = -1. \end{cases}$$

If the down-time associated with any node is greater than the down-time bound of the node, then the δ-constraints for the node are certainly met. The last clause in the equation reflects our assumption that all clocks have a constant value during step -1, *i.e.*, over the interval $[-\infty, t_0]$. The clause could be modified to reflect different assumptions about how a circuit is initialized.

5.3 A verification algorithm for finite clocking schemes

Algorithm FINITE takes a circuit $G = (V, E)$ and a clock set Φ, and verifies in $O((|V|+|E|)K)$ time whether the G operates properly for the first K steps of Φ. Using the base step function $\widehat{B}$, Algorithm FINITE constructs the first K levels of the computational expansion of G and checks the δ-constraints of each node. Only $O(|F|+|E|)$ working space is needed, since nodes in the computational expansion are generated and discarded throughout the course of the algorithm. Total space required is $O(|F|+|E|+|\Phi|K)$, however, since specification of the clocks may require $O(|\Phi|K)$ additional space.

Algorithm FINITE generates, level by level, the nodes in the computational expansion G_{CX}, checking δ-constraints as nodes are generated. The algorithm begins by generating level -1 of G_{CX} and computing for each node the values of $\widehat{B}$ and db. Level 0 of the G_{CX} is then generated using the definitions of $\widehat{B}$ and up-time, and δ-constraints of nodes in level 0 are checked using the definitions of db and down-time. Nodes and their δ-constraints in subsequent levels of G_{CX} are generated and checked in a like manner. Observe, however, that the generation of nodes and constraints in level $(k+1)$ of G_{CX} only requires the quantities $\widehat{B}$ and db for nodes v_i, where $i = \widehat{B}(v,k)$. Consequently, a node v_i in G_{CX} can be discarded, and its storage reused, whenever a new node v_j is generated for a higher level of G_{CX}. Observe, that v_i could not be discarded, if $\widehat{B}$ were not monotone.

Algorithm FINITE

```
1   FOR each component v DO
2       IF v is latch whose clock is LOW during -1st step
3         THEN
4           v.B̂ ← -1
5           v.db ← -∞
6           v.Updated ← TRUE
7         ELSE
8           v.Updated ← FALSE
9   FOR LEVEL = -1 TO K DO
10      FOR each component v DO
11          UPDATE(v, LEVEL)
12      FOR each component v DO
13          IF v is a latch whose clock is HIGH during step
14                LEVEL and is LOW during step LEVEL + 1
15            THEN
16              DownTime ← t_{LEVEL+1}
17              IF DownTime < v.db
18                THEN
19                  TIMING-FAULT ← TRUE
20      FOR each component v DO
21          v.Updated ← FALSE
```

Figure 12: Algorithm FINITE takes a circuit $G = (V, E)$ and a clock set Φ, and verifies in $O((|V|+|E|)K)$ time that G operates properly for the first K steps of Φ.

Routine UPDATE

```
1  IF v.Updated = TRUE or
2       v a latch whose clock is LOW during step LEVEL
2    THEN
3      v.Updated ← TRUE
4      RETURN
5    ELSE
6      FOR each u such that (u, v) ∈ E
7        DO
8          UPDATE(u, LEVEL)
9      IF v is a functional element
10       THEN
11         v.B̂ ← max_{(u,v)∈E} u.B̂
12     ELSEIF v is a latch whose clock is HIGH during step LEVEL and
13               v.B̂ ≥ u.B̂, where (u, v) ∈ E
14       THEN
15         v.B̂ ← u.B̂
16     ELSEIF v is a latch whose clock is HIGH during step LEVEL and
17               v.B̂ < u.B̂ , where (u, v) ∈ E
18       THEN
19         v.B̂ ← LEVEL
20     v.db ← v.d + max_{(v,v)∈E} v.db
21     IF v.db < t_LEVEL and v.B̂ = LEVEL
22       THEN
23         v.db ← t_LEVEL
24     v.Updated ← TRUE
25     RETURN
```

Figure 13: Routine UPDATE routine takes v and $LEVEL$ as arguments and updates the variables $v.\widehat{B}$ and $v.db$ using the definitions for stable configuration and tail weight.

Figure 12 shows Algorithm FINITE. The global variable *LEVEL* indicates the level of G_{CX} currently being worked on, and $v.\widehat{B}$, $v.db$, $v.d$, and $v.Updated$ are fields of a record that holds data for component $v \in V$. The fields $v.\widehat{B}$, $v.db$ and $v.d$ store the base step, down-time bound and propagation delay, respectively, for v during the step corresponding to *LEVEL*, and the variable $v.Updated$ is a flag that indicates whether $v.\widehat{B}$ and $v.db$ have been updated from the values for the previous level of G_{CX}. Lines 1–8 set $v.\widehat{B}$ and $v.db$ to the initial values specified by the definitions of $\widehat{B}$ and down-time bound. Lines 10–19 generate nodes and test δ-constraints for a single level of G_{CX}. The subroutine UPDATE, is shown in Figure 13, and computes $v.\widehat{B}$ and $v.db$ for the level of G_{CX} being worked on. UPDATE is implemented recursively, with a straightforward coding of the recursive definitions of $\widehat{B}$ and *db*.

For each level of G_{CX}, the total time needed to perform all calls to UPDATE is $O(|V| + |E|)$, or $O(|E|)$ if G is connected. To show this, we break the calls to UPDATE into two categories. Calls to UPDATE that terminate because v has already been updated are *cheap*, and calls that actually calculate new values for $v.\widehat{B}$ and $v.db$ are *expensive*. Cheap calls require only constant time and are charged a single unit of time. Expensive calls make recursive calls to UPDATE, and then perform computations that require time proportional to the number of edges to v. The time required for the recursive calls is charged to those calls, while the time required to actually compute $v.\widehat{B}$ and $v.db$ is charged to the call itself. For any component, only a single expensive call is ever made. Consequently, the total time required for all expansive calls is $O(|V| + |E|)$. Similarly, since each component makes at most one cheap call for each input, the total time required for all cheap calls is also $O(|V| + |E|)$.

Algorithm FINITE runs in $O((|V| + |E|)K)$ time and $O(|V| + |E| + |\Phi|K)$ space. Except for the calls to UPDATE, the time bound for the internal loop, Lines 10–21, of Algorithm FINITE is $O(|V|)$. Thus, since the total time needed for all calls to UPDATE can be shown to be $O(|V| + |E|)$ for each level of G_{CX}, the total time needed by Algorithm FINITE is $O((|V| + |E|)K)$. Algorithm FINITE requires storage for a constant number of variables per component in G, the structure of G itself and the clock values for K steps. Thus the total space required is $O(|V| + |E| + |\Phi|K)$.

6 Verifying circuits with periodic clock sets

In this section, we examine how all constraints in Δ can be checked in the practical case of circuits with periodic clock sets. In particular, we describe how, for periodic clock sets, the infinite number of constraints in Δ can be checked in polynomial time. The method partitions the computational expansion into subgraphs, or frames, which are essentially the computational expansions of individual clock periods. Constraints in Δ are then divided into internal constraints that correspond to paths within individual frames, and cross constraints that correspond to paths that include nodes from multiple frames. Violated constraints of either type can be detected by searching for negative weight paths in an augmented copy of a single "pessimistic" frame. The methods we describe immediately lead to an algorithm for verifying the proper operation of periodically clocked circuits.

6.1 Frames of Computational Expansions

The period π of a clock set Φ provides a natural partition of the constraint set Δ. For any time t, if $t = t_0 + j\pi + x$ where j is a nonnegative integer and $0 \leq x < \pi$, then t is in the jth *period* of Φ and x is the *offset* of time t. A Δ-constraint is a *internal* constraint, if the corresponding up-time and down-time of the constraint are both in the same clock period. A Δ-constraint is a *cross* constraint, if the corresponding up-time and down-time of the constraint are in different clock periods. Observe, that since times less than t_0 are not part of any period, Δ-constraints with corresponding up-times of $-\infty$ (*i.e.*, t_{-1}) are technically neither internal constraints nor cross constraints. This boundary condition artifact is of no consequence, however, since Δ-constraints with corresponding up-times of $-\infty$ can never be violated.

The period π of a clock set Φ also provides a natural partition of a computational expansion. We define the kth *contour* of the computational expansion to be all nodes v_i such that $i = B(v, k)$. For any nonnegative integer j, the jth *frame* of the computational expansion is the vertex induced subgraph of G_{CX} containing all nodes in contours (jP) to $(jP + (P - 1))$, where P is the number of steps in the time interval $(t_0, t_0 + \pi]$. Observe that a particular contour may contain nodes from several different levels, and may share nodes with other contours. Internal constraints must correspond to paths that are completely contained within a single frame, while cross constraints must correspond to paths that include nodes from two or more frames.

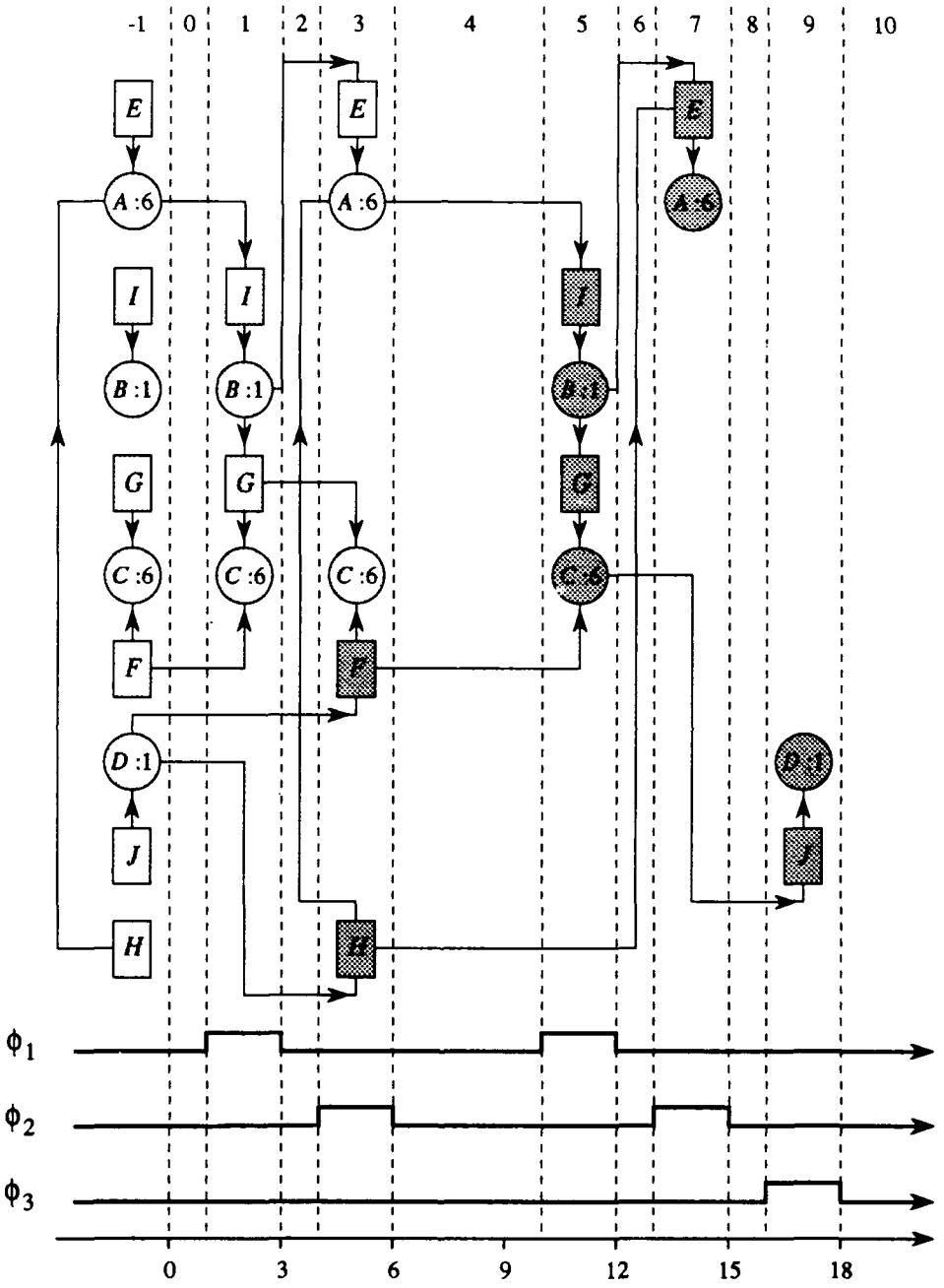

Figure 14: Computational expansion frame for the circuit from Figure 2. The slacks for the frame are listed in Table 1.

Figure 14 shows the first frame of the computational expansion generated by $\widehat{B}$ for the circuit from Figure 2. Nodes in different levels of the computational expansion are separated by dashed lines. All latches in a particular level must have identical up-times, and in this relatively simple example all latches in a particular level also have identical down-times. Consequently, the up-times and down-times of the various latches can be read from the dashed lines that enclose the latch. The 0th contour consists of all the nodes in level -1, while the Pth (*i.e.*, 10th) contour consists of all the darkly shaded nodes. Observe that the 10th contour contains nodes from levels 9, 7, 5, and 3, and shares nodes with contours 3 through 9. In fact, for the example shown, contours $(P-1)$ and P are identical.

The key to checking the infinite number of Δ-constraints, in an amount of time which is polynomial in the size of a given circuit, is the fact that it is sometimes possible to isolate a single "pessimistic" frame of a computational expansion. For convenience, we extend the definitions of "period" and "offset" so that they apply to steps as well as times. For nonnegative integers i, j, and k, if $i = jP + k$ where $k < P$, then step i of Φ is in the jth *period* of Φ and k is the *offset* of step i. Frame j of a computational expansion generated with base step function B is *strict*, if for all components v, periods n, and offsets k,

$$(B(v, jP+k) \neq jP+k) \Rightarrow (B(v, nP+k) \neq nP+k).$$

The equation essentially states that if the base step function for a component v does not change value at a particular offset k into a strict frame, then the base step function for v cannot change value at offset k into any other frame. Intuitively, a frame is strict if component outputs change more frequently during that frame than during any other frame.

An alternate way to define a strict frame would be to require that all components have "more recent" base steps during that frame than during any other frame. Formally, frame j is strict, if for all components v, periods n, and offsets k, $[(nP+k) - B(v, nP+k)] \geq [(jP+k) - B(v, jP+k)]$, *i.e.*, the differences between steps and base steps are minimized during a strict frame. The two definitions are not identical, since frames that are strict by the first definition may not be strict by the alternate definition. Consider, for example, a computational expansion G_{CX} generated with B_{trad}. Since B_{trad} "makes a copy" of a latch each time that the clock of the latch changes value from LOW to HIGH, the fact that the clock set is periodic implies that all frames in G_{CX} are strict by the first definition. Observe, however, that not all frames in G_{CX} are identical. In particular, the first contour of frame 0 is, in general, different from the first contour of any other frame, due to step -1 boundary condition, and in fact only frame 0 would be considered to be strict by the alternate definition. It can be shown, however, that the two definitions are identical, except for such boundary cases involving nodes in the first contour of frame 0. Given the fact that Δ-constraints with corresponding up-times of $-\infty$ can never be violated, the first definition of strictness is preferred for its wider applicability.

For the base step function $\widehat{B}$, frame 0 is strict. In fact, it is possible to show the somewhat stronger property that for all components v, and offsets i, the difference between $(kP+i)$ and $\widehat{B}(v, kP+i)$ can only increase from one frame to the next.

Lemma 6.1 *If Φ is a clock set with P steps in each period, then for any component v and step $k \geq 0$,*

$$\widehat{B}(v,k) \geq \widehat{B}(v,k+P) - P.$$

Proof: The lemma certainly holds for $k = -1$, since for any component v the definition of $\widehat{B}$ states that $\widehat{B}(v,-1) = -1$, and $\widehat{B}(v,-1+P)$ can be at most $P-1$, *i.e.*, less than or equal to $\widehat{B}(v,-1)+P$.

If the lemma holds for all steps less than k, and v is a latch whose clock is LOW during step k, then the lemma holds trivially for v and k. If v is a latch whose clock is LOW during step k, then $\widehat{B}(v,k) = \widehat{B}(v,k-1)$. Similarly, since the clock of v must also be LOW during step $k+P$, $\widehat{B}(v,k+P) = \widehat{B}(v,k+P-1)$. Consequently, $\widehat{B}(v,k) \geq \widehat{B}(v,k+P) - P$, since by assumption $\widehat{B}(v,k-1) \geq \widehat{B}(v,k-1+P) - P$.

Given that the lemma holds at step k for latches whose clocks are LOW during step k, a simple proof-by-contradiction shows that the lemma holds at step k for all components. Assume that the lemma fails to hold for some component at step k. The lemma can only fail to hold for functional elements or latches whose clocks are HIGH during step k. In addition, since we consider only fully synchronous clock sets, there must exist some component v for which the lemma holds for any component u whose output is an input to

v, but fails for v itself. Let v be such a component, and consider the different cases from the definition of $\widehat{B}$. If v is a functional element, then the lemma must hold for v, since the assumption that the lemma holds for any input u, implies that

$$\max_{(u,v)\in E} \widehat{B}(u,k) \geq \left(\max_{(u,v)\in E} \widehat{B}(u,k+P) \right) - P.$$

If v is a latch where either $\widehat{B}(v,k-1) < \widehat{B}(u,k)$ or the clock of v is LOW during steps -1 through $k-1$, then $\widehat{B}(v,k) = k$, and thus, since $\widehat{B}(v,k+P)$ can be at most $k+P$, we have $\widehat{B}(v,k) \geq \widehat{B}(v,k+P) - P$ and the lemma must hold for v. If v is a latch where $\widehat{B}(v,k-1) \geq \widehat{B}(u,k)$, then there are two subcases to consider. The first is when $\widehat{B}(v,k-1+P) \geq \widehat{B}(u,k+P)$. In this case, one can show that the lemma holds for v, since $\widehat{B}(v,k) = \widehat{B}(v,k-1)$, $\widehat{B}(v,k+P) = \widehat{B}(v,k-1+P)$, and $\widehat{B}(v,k-1) \geq \widehat{B}(v,k-1+P) - P$. The second subcase is when $\widehat{B}(v,k-1+P) < \widehat{B}(u,k+P)$. Here, a formal proof is somewhat involved but the general strategy is to show that no v can fall into this subcase. More specifically, it is possible to show that if $\widehat{B}(v,k-1+P) < \widehat{B}(u,k+P)$, then either the clock of v is LOW for some interval of time $\langle t, t_{k+P} \rangle$ which includes step $\widehat{B}(u,k+P)$, or $\widehat{B}(u,k+P) = k+P$. If $\widehat{B}(u,k+P) = k+P$, then we can show that $\widehat{B}(v,k-1) \geq k$, which is clearly absurd. If the clock of v is LOW for some interval of time $\langle t, t_{k+1} \rangle$, which includes step $\widehat{B}(u,k+P)$, then the assumption that $\widehat{B}(u,k) \geq \widehat{B}(u,k+P) - P$ implies that $\widehat{B}(v,k-1) < \widehat{B}(u,k)$, contradicting the premise that $\widehat{B}(v,k-1) \geq \widehat{B}(u,k)$. ∎

The proof of Lemma 6.1 requires the assumption, mentioned in Section 5.1, that the output of a latch whose clock is initially LOW, always changes value the first time that the clock for the latch becomes HIGH. The assumption provides a basis for the induction used to prove Lemma 6.1 and in turn makes the identification of a strict frame simple. If the assumption were removed and replaced with a specification of initial base steps for components, then a strict frame may be difficult to identify, and, indeed, may not even exist. In such cases, however, it is generally possible to construct a strict pseudoframe, *i.e.*, one that is more strict than any actual frame in the computational expansion, but that does not itself exist in the computational expansion. Of course, the generally "pessimistic" nature of such a pseudoframe may lead to the disqualification of some types of properly operating circuits.

6.2 Internal Constraints

To check all internal constraints, it is sufficient to just check the internal constraints of a single strict frame. Given Lemma 6.1, the following theorem essentially states that any violated internal constraint of a computational expansion generated with $\widehat{B}$ can be detected by checking the internal constraints of frame 0.

Theorem 6.2 *If a frame of a monotone computational expansion is strict, then all internal constraints for all frames are met if and only if the internal constraints for the strict frame are met.*

Proof: Let frame j be strict. All internal constraints are met only if the internal constraints in frame j are met, since the internal constraints in frame j are in fact internal constraints in the computational expansion. In addition, since frame j is strict, an argument similar to the one used to prove Lemma 5.1 shows that all internal constraints in the original computational expansion met whenever all internal constraints in frame j are met. ∎

6.3 Cross Constraints

Strict frames can also be used to check cross constraints. A cross constraint corresponds to a path σ which includes nodes from multiple frames of the computational expansion. Observe, that all the nodes from a particular frame i form a subpath σ_i of σ, and each σ_i contributes some delay to the cross constraint, while intuitively, the fact that σ_i includes nodes from different contours of the computational expansion implies that σ_i contributes some time to the "down-time to up-time" part of the cross constraint. The first subpath of σ contributes its delay, and the amount of time between the up-time of the first node in the path and the end of the clock period containing the up-time. The last subpath of σ contributes its delay, and the amount of time between the down-time of the last node in the path and the start of the clock period containing the down-time. Other subpaths contribute their delay, and the amount of time contained in a full clock period. By summing all the contributed delays and times, we can obtain the complete cross constraint.

Given a frame that begins with contour k, the slacks for the frame encode the worst case delay-and-time contributions of subpaths in the frame. The clock period π_k associated with the frame is the interval $\langle t_k, t_{k+P}\rangle$, and is of length π. For any node v_i in first contour of the frame, if σ is a path from v_i to any latch in the frame such that the quantity $(t - t_k) - d(\sigma)$ is minimized, where $t \in \pi_k$ is a down-time associated with the latch, then the quantity $head(v) \stackrel{\text{def}}{=} (t - t_k) - d(\sigma)$ is the *head slack* of the component corresponding to v_i. Similarly, for any node u_j in the first contour immediately after the frame, if σ is the path from any latch in the frame such that the quantity $(t_{k+P} - t) - (d(\sigma) - d(u_j))$ is minimized, where $t \in \pi_k$ is a up-time associated with the latch, then the quantity $tail(u) \stackrel{\text{def}}{=} (t_{k+P} - t) - (d(\sigma) - d(u_j))$ is the *tail slack* of the component corresponding to u_j. Finally, for any node v_i, in first contour of the frame, and node u_j, in the first contour immediately after the frame, if σ is a path from v_i to u_j such that the quantity $\pi - (d(\sigma) - d(u_j))$ is minimized, then the quantity $frame(v, u) \stackrel{\text{def}}{=} \pi - (d(\sigma) - d(u_j))$ is the *frame slack* of the pair of components corresponding to v_i and u_j. If no path exists between v_i and u_j, then there is no frame slack for the pair (v, u). Similarly, no head slack (or tail slack) exists for component v if no paths exist to (or from) nodes with associated down-times (or up-times). Intuitively, the slacks are the worst-case differences between the available amount of time for nodes along a path in the frame to compute and the amount of time that they require.

$v \in V$	Frame Slacks ($frame(v,u)$)										Head Slacks	Tail Slacks
	A	B	C	D	E	F	G	H	I	J	$head(v)$	$tail(v)$
A	4	5	4	-2	4	-	4	-	5	-2	-4	5
B	-	-	-	-	-	-	-	-	-	-	-	8
C	-	-	-	-	-	-	-	-	-	-	-	7
D	10	11	10	4	10	17	10	17	11	4	4	1
E	4	5	4	-2	4	-	4	-	5	-2	-4	7
F	-	-	-	-	-	-	-	-	-	-	-	-
G	-	-	-	-	-	-	-	-	-	-	-	7
H	4	5	4	-2	4	-	4	-	5	-2	-4	-
I	-	-	-	-	-	-	-	-	-	-	-	8
J	10	11	10	4	10	17	10	17	11	4	4	1

Table 1: Frame slacks, head slacks, and tail slacks for the frame shown in Figure 14.

The slacks for the frame shown in Figure 14 can be read from Table 1. For example, the table states that $head(A) = -4$. Referring back to Figure 14, it is apparent that this is indeed the case, since G_1 has a down-time of 3, A_{-1} is in the first contour of the frame, and there exists a path $\sigma = A_{-1} \rightarrow I_1 \rightarrow B_1 \rightarrow G_1$ from A_{-1} to G_1. A search of the frame demonstrates that σ is a worst-case path, and thus $head(A) = (3 - t_0) - d(\sigma) = 3 - 7 = -4$. Similarly, the table states that $tail(A) = 5$. Here, the worst-case path is $\sigma' = E_7 \rightarrow A_7$, and thus $tail(A) = (t_{10} - t_7) - (d(\sigma') - d(A_7)) = (18 - 13) - (6 - 6) = 5$. Observe, that while A_7 is part of the shown frame, A_7 is also in the first contour of the next frame, *i.e.*, the 10th contour of the complete computational expansion. As a final example, the table states that $frame(A, D) = -2$. The worst-case path is $\sigma'' = A_{-1} \rightarrow I_1 \rightarrow B_1 \rightarrow E_3 \rightarrow A_3 \rightarrow I_5 \rightarrow B_5 \rightarrow G_5 \rightarrow C_5 \rightarrow J_9 \rightarrow D_9$, and thus $frame(A, D) = \pi - (d(\sigma'') - d(D_9)) = 18 - 20 = -2$.

An interesting feature of frame shown in Figure 14 is that a large number of slacks do not exist, as indicated by the dashes in Table 1. For example, no slacks at all are shown for latch F. This is not surprising, given the structure of the frame. Observe, that no paths in the frame lead from F_{-1} (*i.e.*, the copy of F in the first contour of the frame) to any other latches. Consequently, no head slack exists for F. Also, there is no path in the frame from F_{-1} to any component in the first contour of the next frame, so no frame slacks exist for F. Finally, since the up-time associated with J_{-1} is outside the clock period for the frame, and $J_{-1} \rightarrow D_{-1} \rightarrow F_3$ is the only path from some other latch in the frame to the copy F_3 of F in the first contour of the next frame, no tail slack exists for F.

If there exists a strict frame, then all cross constraints can be checked by examining sequences of slacks. Let $s = v, u, w, \ldots, x, y$ be any sequence of components, possibly with more than a single occurrence of a particular component. If the slacks of frame j are such that

$$tail(v) + frame(v, u) + frame(u, w) + \ldots + frame(x, y) + head(y) < 0,$$

then s is a *negative slack sequence* for frame j.

Theorem 6.2 *If a cross constraint in a monotone computational expansion is violated, and there exists a strict frame, then either the strict frame contains a violated internal constraint or there exists a negative slack sequence for the frame.*

Proof: The theorem can be proved with methods similar to those used in Theorem 6.1, but making use of a new base step function B_j. If frame j of a computational expansion generated with some monotone base step function B is strict, then the base step function B_j essentially specifies for every component step pair $(v, nP+i)$ a base step l such that the number of steps between $nP+i$ and l is always the same as the number of steps between step $(jP+i)$ and the base step for v at step $(jP+i)$, specified by B. Thus, the computational expansion generated with B_j is essentially the computational expansion that would result if every frame "looked like" frame j. Formally, for any period n, let $x(n) = nP - jP$. Now, for any offset i, and component v,

$$B_j(v, nP+i) = \begin{cases} B(v, jP+k) & \text{if } n = j \\ x(n) + \max_{0 \le k \le i} B(v, jP+k) & \text{if } \exists k, \text{ such that } 0 \le k \le i \text{ and } B(v, jP+k) = jP+k \\ x(n) - P + \max_{0 \le k < P} B(v, jP+k) & \text{otherwise.} \end{cases}$$

In order to preserve initial conditions, $B_j(v,-1) = B(v,-1)$ for any component v. It is tempting to think that $B_j(v, nP+i)$ could be defined as $x(n) + B(jP+i)$. The problem, not surprisingly, is with boundary conditions. For example, if $B = \widehat{B}$, and $j = 0$, the naive definition would specify that $B_j(v,P) = P-1$ for all v. This specification would probably be inconsistent, since for some v it is almost certain that $B(v, P-1) \neq P-1$.

Unfortunately, if frame j is not the same as frame 0, then the definition of B_j might reference nonexistent steps that are "before" step 0. Such references to nonexistent steps can be resolved by adding a finite number of suitable additional steps before step 0. Let level i be the earliest level that contains nodes in frame j. Level i corresponds to step $\langle t_i, t_{i+1} \rangle$, so we can add the needed steps by replacing each clock ϕ with an augmented clock ϕ', that is defined as follows:

$$\phi'(t) = \begin{cases} \phi(t) & \text{if } t \ge t_0, \\ \phi(-\infty) & \text{if } t \in [-\infty, t_0 - (t_{jP} - t_i)), \\ \phi(t+\pi) & \text{if } t \in \langle t_0 - (t_{jP} - t_i), t_0]. \end{cases}$$

Observe, that the number of additional steps in ϕ' must be less than the number of steps in a single period of ϕ, or else frame j could certainly not have been strict. By assuming the augmented clocks, and using arguments similar to those in the proof to Lemma 3.1, B_j can be shown to be an expanding monotone base step function which by Lemma 5.1 is more strict than the original base step function B.

The fact that slacks correspond to minimal time-minus-delay pairs, implies that whenever there exists a violated cross constraint in the computational expansion G_{CX_j} generated by B_j, there must also exist a negative slack sequence for frame j. The path σ in G_{CX_j} that corresponds to the violated constraint, can be broken into subpaths $\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n$, where each subpath only contains nodes from successive frames. Let $v^{(i)}$ denote the first node in σ_i. If $v^{(n)}$ is a copy of component u, then the definition of head slack guarantees that $head(u) \le t_{down} - d(\sigma_n)$, where t_{down} is the amount of time between the start of the clock period associated with σ_n, and the down-time associated with the last node in σ_n. Similarly, if $v^{(2)}$ is a copy of component w, then the definition of tail slack guarantees that $tail(w) \le t_{up} - d(\sigma_1)$, where t_{up} is the amount of time between the the up-time of $v^{(1)}$ and the end of the clock period associated with σ_1. Finally, for $i = 2, 3, \ldots, n-1$, if $v^{(i)}$ is a copy of component u and $v^{(i+1)}$ is a copy of component w, then the definition of frame slack guarantees that $frame(u,w) \le \pi - d(\sigma_i)$. The fact that σ corresponds to a constraint violation implies that the sum of the quantities $t_{up} - d(\sigma_1)$, $\pi - d(\sigma_2)$, ..., $\pi - d(\sigma_{n-1})$ and $t_{down} - d(\sigma_n)$ must be negative, and thus, $v^{(2)}, v^{(3)}, v^{(4)}, \ldots, v^{(n)}$ is a negative slack sequence .

The remainder of the proof uses arguments similar to those in Lemma 5.1. As in the proof of Lemma 5.1, it can be shown that a violated cross constraint in the original computational expansion G_{CX} implies a

violated constraint in the computational expansion G_{CX_j} generated by B_j. Let σ be the path corresponding to the violated constraint in G_{CX}. By "backtracking" through the two computational expansions, a path σ' in G_{CX_j} can be constructed, where by the monotonicity of B and B_j, σ' also corresponds to a violated constraint. Now, if σ' corresponds to an internal constraint, then the definition of B_j implies that frame j also contains a violated internal constraint. Also, if σ' corresponds to a cross constraint, then there exists a negative slack sequence for frame j. ∎

Lemma 6.1 implies that Theorem 6.2 can be applied to computational expansions generated with $\widehat{B}$. In addition, an inductive argument similar to that used to prove Lemma 5.1 can be used to show that the converse of Theorem 6.2 holds for the special case of $\widehat{B}$. Unfortunately, the converse of Theorem 6.2 is not true in general, and consequently, there exist base step functions where a negative slack sequence may exist even when no Δ-constraint in the computational expansion is violated. Even in such cases, however, the timing analysis based on the slacks is "safe" in the sense that the presence of a violated Δ-constraint is never overlooked.

Negative slack sequences can be detected using an augmented copy of a frame. Given a circuit $G = (V, E)$, a corresponding computational expansion $G_{CX} = (V_{CX}, E_{CX})$, and a frame $G_F = (V_F, E_F)$ of G_{CX}, let contour k be the first contour in G_F. The *Δ-constraint graph* for the frame G_F is the graph $G_\Delta = (V_\Delta, E_\Delta)$, where

$$
\begin{aligned}
V_\Delta \;=\;& \{v_i : v_i \in V_F\} \cup \{v_\pi : v \in V\} \cup \{s, t\} \cup \\
& \{v_i^{t_{up}} : v_i \in V_F \text{ with up-time } t_{up}\} \cup \\
& \{v_i^{t_{dn}} : v_i \in V_F \text{ with down-time } t_{dn}\}, \\
E_\Delta \;=\;& \{(v_i, u_j) : (v_i, u_j) \in E_F\} \cup \\
& \{(v_i, v_i^{t_{dn}}) : v_i^{t_{dn}} \in V_\Delta\} \cup \\
& \{(v_i^{t_{up}}, v_i) : v_i^{t_{up}} \in V_\Delta\} \cup \\
& \{(v_\pi, v_i) : v_i \text{ is in contour } k \text{ of } G_{CX}\} \cup \\
& \{(u_j, v_\pi) : u_j \text{ is in contour } (k + (P-1)) \text{ of } G_{CX}, \\
& \qquad (u_j, v_l) \in E_{CX} \text{ and } v_l \text{ is in contour } (k + P) \text{ of } G_{CX}\} \cup \\
& \{(v_i^{t_{dn}}, t) : v_i^{t_{dn}} \in V_\Delta\} \cup \\
& \{(s, v_i^{t_{up}}) : v_i^{t_{up}} \in V_\Delta\}.
\end{aligned}
$$

Each $v_i \in V_F$ has a propagation delay equal to $-d(v)$, each v_π has a propagation delay equal to π, each $v_i^{t_{up}}$ has a propagation delay equal to $-(t_{up} \bmod \pi)$, each $v_i^{t_{dn}}$ has a propagation delay equal to $(t_{dn} \bmod \pi)$ and both s and t have propagation delays equal to 0.

The Δ-constraint graph G_Δ has been constructed so that the propagation delays along certain paths are equal to the slacks of the original frame j. For example, if $head(v)$ exists, then there must exist a path σ in frame j, from v_i in the first contour of frame j to some latch u_l, with down time t_m, such that $(t_m - (\pi \cdot j)) - d(\sigma)$ is exactly equal to $head(v)$. Thus, since σ must also exist in G_Δ, and the delay of $u_l^{t_m}$ is defined to be $(t_m \bmod \pi)$, which in turn equals $(t_m - (\pi \cdot j))$, the total delay along the path formed by appending the edge $(u_l, u_l^{t_m})$ onto the end of σ must be equal to $head(v)$. For similar reasons, if $tail(v)$ exists, then there must exist a path σ in G_Δ from some $u_i^{t_n}$ to some v_π, such that $d(\sigma)$ is exactly equal to $tail(v)$. Finally, if $frame(v, u)$ exists, then there must exist in G_Δ a path $\sigma = v_i \xrightarrow{*} u_\pi$ from v_i in the first contour of frame j to u_π such that $d(\sigma) = frame(v, u)$. A path in G_Δ whose total propagation delay must, by construction, be equal to some slack is a *slack path.*

If there exists a strict frame, then all Δ-constraints can be checked by running any of the standard shortest paths algorithms on the Δ-constraint graph G_Δ for the strict frame. A constructive argument can be used to show that if there exists a negative slack sequence, then G_Δ contains a negative-weight path from s to t. In addition, violated internal constraints also imply negative-weight paths from s to t. Thus, by Theorems 6.2 and 6.1, all Δ-constraints can be checked by finding the least-weight path from s to t and comparing the weight of that path to 0.

Theorem 6.3 *If a Δ-constraint in a monotone computational expansion is violated, and frame j is strict, then the Δ-constraint graph G_Δ for frame j contains a negative-weight path from s to t.*

Proof: Consider first the case of a violated internal constraint $t_k - t_i \geq d(\sigma)$, where σ is a path from v_i to u_j within some frame, v_i has up-time t_i, and u_j has down-time t_k. By Theorem 6.1, there must also exist a violated internal constraint $t_n - t_l \geq d(\sigma')$, where σ' is a path from v_l to u_m within the strict frame, v_l has up-time t_l, u_m has down-time t_n, and $d(\sigma') = d(\sigma)$. Now, G_Δ must, by definition, also contain the

path σ', but where the weight of σ' is equal to $-d(\sigma)$. (Recall, that propagation delays are negated in the definition of G_Δ.) In addition, G_Δ must contain a path $s \rightarrow v_l^{t_l} \rightarrow v_l$, whose weight is $-(t_l \bmod \pi)$, and a path $u_m \rightarrow u_m^{t_n} \rightarrow t$, whose weight is $(t_n \bmod \pi)$. Now, all three of these paths, can be combined to form a single path from s to t whose weight is $d(\sigma') + (t_n \bmod \pi) - (t_l \bmod \pi)$. Since the constraint is internal, however, the quantity $((t_n \bmod \pi) - (t_l \bmod \pi)) = (t_n - t_l)$, and thus the fact that the constraint is violated directly implies that the weight of the combined path is less than 0.

The argument for violated cross constraints is similar to the argument for internal constraints. First, if there exists a violated cross constraint, then by Theorem 6.2, either the strict frame contains a violated internal constraint, or the there exists a negative slack sequence. It has already been shown that violated internal constraints imply a negative-weight path from s to t, so all that remains to be shown is that the existence of a negative slack sequence also implies a negative-weight path from s to t.

The final step of the proof is to show that the slack paths corresponding to a negative slack sequence can be combined into a single negative-weight path from s to t. Let v, u, w, ..., x, y be the implied negative slack sequence. Since $tail(v)$ exists, there must exist a slack path σ in G_Δ from some $u_i^{t_{v\pi}}$ to some v_π, such that $d(\sigma)$ is exactly equal to $tail(v)$. In addition, since $frame(v, u)$ exists, there must exist in G_Δ a slack path $\sigma' = v_i \stackrel{*}{\rightarrow} u_\pi$ from v_i in the first contour of frame j to u_π such that $d(\sigma') = frame(v, u)$. Since v_i is in the first contour of frame j, however, E_Δ contains the edge (v_π, v_i), and thus σ and σ' can be combined into a single path whose total weight is equal to $tail(v) + frame(v, u)$. Continuing in this fashion, a path from s to t can be constructed whose total weight is equal to

$$tail(v) + frame(v, u) + frame(u, w) + \ldots + frame(x, y) + head(y).$$

Thus, since this quantity is known to be negative, G_Δ contains a negative-weight path from s to t. ∎

One difficulty with Theorem 6.3 is that it puts no bound on the length of the implied negative-weight path from s to t. This is not surprising, since the negative-weight path corresponds directly to the presumed violated Δ-constraint. Indeed, since there exist circuits, with as few as 4 components, that can operated for an arbitrary, but not infinite, number of clock cycles before some latch fails to hold its proper value, it is certain that the length of the implied path is essentially independent of the size of the circuit. The lack of a useful bound on the length of the implied path indicates that a brute force search for a negative weight path from s to t would not be an efficient way to perform timing verification. Fortunately, the following related lemma helps us out of this difficulty.

Lemma 6.2 *If G_Δ contains a negative weight path σ from s to t, then G_Δ contains a path σ' of length less than $|V|P$, such that σ' is either from s to t and of negative weight, or from s to some other vertex and contains a negative weight cycle.*

Proof: If σ is of length less than $|V|P$, then $\sigma' = \sigma$. If σ is of length greater than or equal to $|V|P$, then one or more vertices must appear more than once in σ, and thus σ must contain one or more cycles. Clearly, if all positive-weight cycles are removed from σ, the resulting path σ'' is still from s to t and has negative weight. If σ'' is of length less than $|V|P$, then $\sigma' = \sigma''$. Otherwise, the path σ''' formed by taking the first $|V|$ edges in σ'' must itself contain a cycle. In addition, any cycle in σ''' must have negative weight, since σ'' contains no positive cycles, so $\sigma' = \sigma'''$. ∎

Lemma 6.2 is essentially the last step in an argument stating that a standard shortest-path algorithm can be used to check all the Δ-constraints of a given circuit. Theorems 6.1 and 6.2 stated that any violated Δ-constraint could be detected by examining a single strict frame, while Lemma 6.1 confirmed that a strict frame existed for computational expansions generated with $\widehat{B}$. Theorem 6.3 then showed that the necessary "examination" of the strict frame could be performed by finding the shortest path between two nodes in the Δ-constraint graph for a strict frame, while Lemma 6.2 showed that in fact the general search for a negative-weight path could be replaced with a search for "short" negative-weight paths of a particular type. Thus, the lemma completes the argument, since the Bellman-Ford shortest-path algorithm [12] can be used to detect paths of precisely this type.

6.4 A verification algorithm for circuits with periodic clock sets

Algorithm PERIODIC takes a circuit $G = (V, E)$ and a periodic clock set Φ and verifies the proper operation of G in $O(|V||E|P)$ time and $O((|V| + |E| + |\Phi|)P)$ space, where P is the number of steps in a single period of Φ. Since P is generally a small constant, the time and space requirements of Algorithm PERIODIC are

Algorithm PERIODIC

1. Construct frame 0 of the computational expansion.
2. Modify frame 0 to obtain the Δ-constraint graph for frame 0.
3. Compute the shortest path from s to t.
4. Check Δ-constraints by comparing 0 to the shortest path from s to t.

Figure 15: Algorithm PERIODIC verifies the proper operation of a circuit $G = (V, E)$, with periodic clock set Φ, in $O((|V||E|P)$ time and $O((|V| + |E| + |\Phi|)P)$ space, where P is the number of steps in a single period of Φ.

essentially $O(|V||E|)$ and $O(|V| + |E|)$, respectively. Algorithm PERIODIC uses the results of Theorem 6.3 and Lemma 6.1 to check the Δ-constraints for G. The time intensive part of Algorithm PERIODIC involves detecting negative-weight paths in the Δ-constraint graph of a strict frame.

Figure 15 shows a high-level statement of Algorithm PERIODIC. Given a level-clocked circuit $G = (V, E)$ and a clock set with P steps per period, Algorithm PERIODIC checks the Δ-constraints that would result if the P steps of the clocks were repeated indefinitely. All Δ-constraints are checked by constructing the Δ-constraint graph for frame 0 and then using a single-pair shortest-path algorithm to check for a negative-weight path from s to t. By Theorem 6.3, Algorithm PERIODIC checks all Δ-constraints.

The bulk of the time required by Algorithm PERIODIC goes to computing the shortest path from s to t in the Δ-constraint graph for frame 0. The construction of frame 0 and its Δ-constraint graph can be performed in $O((|V| + |E|)P)$ time and $O(|V| + |E| + |\Phi|P)$ space, with a subroutine similar to Algorithm FINITE. To check for negative-weight paths from s to t, the Bellman-Ford shortest-path algorithm [12] can be used. The Bellman-Ford algorithm can detect paths of the type specified by Lemma 6.2 in $O(|V_\Delta||E_\Delta|)$ time and $O(|V_\Delta|+|E_\Delta|)$ space. Observe, however, that $|V_\Delta|$ is proportional to $|V|P$, and $|E_\Delta|$ is proportional to $|E|P$, so the time and space bounds can be restated as $O(|V||E|P^2)$ and $O((|V| + |E|)P)$, respectively.

Using a modified version of the Bellman-Ford algorithm, however, the total running time of Algorithm FINITE can be reduced to $O(|V||E|P)$. The standard Bellman-Ford algorithm can be conceptually viewed as a series of "relaxation" steps, where each relaxation examines every edge exactly once. The number of relaxations required depends on the order in which edges are examined, but in the worst case each relaxation only determines one edge in the "shortest path" being sought, so $|V|P$ relaxations are needed for the path σ' implied by Lemma 6.2. Thus, since there are $|E|P$ edges to examine, the total running time for the standard Bellman-Ford algorithm would be $O(|V||E|P^2)$. It is possible to show, however, that there exists another path, analogous to σ', that can be found in $|V|$ relaxations, if edges are examined in a special order.

The new required new path can be shown to exist using arguments similar to those in the proof of Lemma 6.2. In the proof of Theorem 6.3, it was shown that the negative weight path implied by the theorem could be broken into subpaths, where each subpath corresponded to a slack. In addition, all of these *slack paths* have less than $|V|P$ edges, and all but the last ends on some vertex v_π. Now, by repeating the argument from the proof of Lemma 6.2, but only allowing the removal of complete slack paths, it is not difficult to argue that there exists a path σ'' consisting of $|V|$ or fewer slack paths, such that either σ'' is from s to t and of negative total weight, or σ'' is from s to some v_π and contains a negative weight cycle that includes some u_π.

The Bellman-Ford algorithm can detect the existence of σ'' in $|V|$ relaxations, if the edges in E_Δ are examined in "topological" order [3, 19]. Technically, since G_Δ is not acyclic, no true topological order exists for the edges in E_Δ. Fortunately, it is possible to obtain an ordering which is essentially topological, by removing from E_Δ all edges from v_π vertices, topologically ordering all the edges that remain, and then appending to the topological ordering the edges from v_π vertices. The specific ordering of the edges from v_π vertices is unimportant. Now, since no slack path can contain an edge from a v_π vertex, examining the remaining edges in the order that they appear topologically allows the Bellman-Ford algorithm to "find" an entire slack path in each relaxation, rather than just a single edge. Thus, since σ'' consists of at most $|V|$ slack paths, only $|V|$ relaxations are needed, and algorithm terminates in $O(|V||E|P)$ time.

Overall, therefore, Algorithm PERIODIC runs in $O(|V||E|P)$ time and $O((|V| + |E| + |\Phi|)P)$ space. In practice, both P and $|\Phi|$ are small constants, so our time and space bounds become $O(|V||E|)$ and $O(|V| + |E|)$, respectively. In addition, the fanout of actual circuit components is generally restricted to a small constant, so we expect that for large circuits $|E|$ is roughly proportional to $|V|$, and our time bound becomes $O(|V|^2)$. As a practical matter, it may be possible to obtain nearly linear observed running times, by computing the shortest paths using the SHORTESTPATHTREE algorithm presented by Tarjan [19, p. 92].

When edge weights are integers, efficient "scaling" algorithms can be used to solve the single-source shortest-paths problem. For graphs with negative edge weights, the algorithm due to Gabow and Tarjan [5] runs in $O(\sqrt{V}E\lg(VW))$ time, where W is the magnitude of the largest magnitude weight of any edge in the graph.

Often in practice, multiple clocks are derived from a single fundamental clock, and it is of great interest to know the maximum frequency of the fundamental clock. It is also desirable to know the critical path that limits this frequency. By using methods similar to those used to solve the Minimal Cost-to-Time Ratio Cycle Problem [12], our algorithm can be adapted to determine, in polynomial time, the maximum clock frequency and its associated critical path.

In order to test the real-world practicality of Algorithm PERIODIC, an experimental timing verification tool CXCLONE is currently under development. Once completed, CXCLONE will be used to test the observed efficiency of Algorithm PERIODIC on a variety of solicited academic and industrial VLSI circuits. In addition, by incorporating mechanisms for handling standard design activities such as "false path" disabling and hierarchical circuit analysis CXCLONE will be used to explore how the methods presented in this paper can be applied in an integrated design environment.

7 Conclusion

This paper has established a formal framework for understanding level clocking in VLSI systems. A key idea in the framework is the use of a base step function to capture any particular set of timing assumptions about "when things change." The computational expansion of a circuit depends on the results of the base step function, but not on the details of how those results are computed. Thus, our methodology for verifying circuits applies equally well to any set of timing assumptions that can be expressed in terms of a base step function, not just the $\widehat{B}$ function presented.

The framework can be extended to address many design concerns. For example, the framework can be extended to handle noninstantaneous clock transitions by using a somewhat more complex circuit model [10], and making suitable modifications to the definitions of $\widehat{B}$, up-time and down-time. Global clock skew on a chip can be handled in a similar fashion. Set-up times for latches can also be checked using modifications to the definition of down-time, but the checking of hold times for latches is more problematic, since we do not model the nonzero minimum propagation delays that are frequently used to satisfy hold time requirements. We have also made some preliminary studies of circuits incorporating multiplexors whose control inputs are periodic, and it appears that our framework can be used to analyze these circuits as well.

So-called "two-sided" timing constraints, in which functional elements have minimum propagation delays, are more problematic. For the "one-sided" constraints we have considered, the designer's intent can be inferred by letting propagation delays go to 0. For circuits designed with two-sided constraints, the isolation of ideal outputs is more difficult. We are currently working on the problem of verifying circuits with two-sided constraints u.ing the notions of base step functions and computational expansion.

Some timing analyzers attempt to handle circuits with data-dependent delays: propagation delays of functional elements that depend on the particular values of inputs to the element. Our method of computational expansion applies perfectly well to the analysis of such circuits, but the base step function $\widehat{B}$ used by our algorithms is, unfortunately, not sophisticated enough to cope with data-dependent delays. Whether an efficiently computable base step function can be developed for this situation is an open research question.

References

[1] V. D. Agrawal, "Synchronous path analysis in MOS circuit simulator," in *Proc. 19th ACM/IEEE Design Automation Conference* (1982), pp. 629–635.

[2] A. G. Bose, and K. N. Stevens, *Introductory Network Theory*, Harper and Row, 1965.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press and McGraw-Hill, 1990.

[4] M. R. Daganais and N. C. Rumin, "Automatic determination of optimal clocking parameters in synchronous MOS VLSI circuits," in *Advanced Research in VLSI: Proc. of the 5th MIT Conference* (1988), 19–33.

[5] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for network problems," *SIAM Journal on Computing*, Vol. 18, No. 5, October 1989, pp. 1013–1036.

[6] L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley, Reading, Massachusetts, 1985.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Co., San Francisco, 1979.

[8] M. Glesner, J. Schuck, and R. B. Steck, "SCAT—a new statistical timing verifier in a silicon compiler," in *Proc. 23rd ACM/IEEE Design Automation Conference* (1986), pp. 220–226.

[9] R. B. Hitchcock, Sr., "Timing verification and the timing analysis program," in *Proc. 19th ACM/IEEE Design Automation Conference* (1982), pp. 594–604.

[10] A. I. Ishii, *A Digital Model for Level-Clocked Circuitry*, Masters thesis, Laboratory of Computer Science, Massachusetts Institute of Technology, 1988.

[11] N. P. Jouppi, *Timing Verification and Performance Improvement of MOS VLSI Designs*, Ph.D. dissertation, Computer Systems Laboratory, Stanford University, 1984. Also available as Technical Report No. 84-266.

[12] E. L. Lawler, *Combinatorial Optimizaton: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[13] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Massachusetts, 1980.

[14] T. M. McWilliams, "Verification of timing constraints on large digital systems," in *Proc. 17th ACM/IEEE Design Automation Conference*, 1980, pp. 139–147.

[15] M. Muraoka, H. Iida, H. Kikuchihara, M. Murakami, annd K. Hirakawa, "ACTAS: An accurate timing analysis system for VLSI," in *Proc. 22 nd ACM/IEEE Design Automation Conference*, 1985, 152–158.

[16] J. K. Ousterhout, "A switch-level timing verifier for digital MOS VLSI," *IEEE Transactions on Computer-Aided Design*, CAD-4, No. 3, July 1984, pp. 336–349.

[17] K. A. Sakallah, T. N. Mudge and O. A. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits," CSE-TR-31-89, Computer Science and Engineering Division, University of Michigan, Ann Arbor, Michigan, 1989.

[18] T. G. Szymanski, "LEADOUT: A static timing analyzer for MOS circuits," in *Proc. 1986 IEEE International Conference on CAD*, 1986, pp. 130–133.

[19] R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1983.

[20] Texas Instruments Incorporated, *The TTL Data Book for Design Engineers*, Dallas, Texas, 1976.

[21] S. H. Unger and C. J. Tan, "Clocking schemes for high speed digital systems," *IEEE Transactions on Computers*, Vol. C-35, No. 10, October 1986, pp. 880–895.

[22] N. Weiner and A. Sangiovanni-Vincentelli, "Timing analysis in a logic synthesis environment," in *Proc. 26th ACM/IEEE Design Automation Conference*, 1989, pp. 655–661.

DARPA OFFICIAL DISTRIBUTION LIST

DIRECTOR — 2 copies
Information Processing Techniques Office
Defense Advanced Research Projects Agency (DARPA)
1400 Wilson Boulevard
Arlington, VA 22209

OFFICE OF NAVAL RESEARCH — 2 copies
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. Gary Koop, Code 433

DIRECTOR, CODE 2627 — 6 copies
Naval Research Laboratory
Washington, DC 20375

DEFENSE TECHNICAL INFORMATION CENTER — 2 copies
Cameron Station
Alexandria, VA 22314

NATIONAL SCIENCE FOUNDATION — 2 copies
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

HEAD, CODE 38 — 1 copy
Research Department
Naval Weapons Center
China Lake, CA 93555